This Page Is Inserted by IFW Operations
and is not a part of the Official Record

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images may include (but are not limited to):

- BLACK BORDERS
- TEXT CUT OFF AT TOP, BOTTOM OR SIDES
- FADED TEXT
- ILLEGIBLE TEXT
- SKEWED/SLANTED IMAGES
- COLORED PHOTOS
- BLACK OR VERY BLACK AND WHITE DARK PHOTOS
- GRAY SCALE DOCUMENTS

IMAGES ARE BEST AVAILABLE COPY.

As rescanning documents *will not* correct images, please do not report the images to the Image Problem Mailbox.

METHOD FOR RECOGNIZING CHARACTER AND DEVICE THEREFOR

Patent number: JP6068301

Publication date: 1994-03-11

Inventor: WANG SHIN-YWAN; VAEZI MEHRZAD R; SHERRICK CHRISTOPHER A

Applicant: CANON INC;; CANON INF SYST INC

Classification:

- international: G06K9/20; G06K9/32; G06K9/34

- european:

Application number: JP19930121883 19930426

Priority number(s):

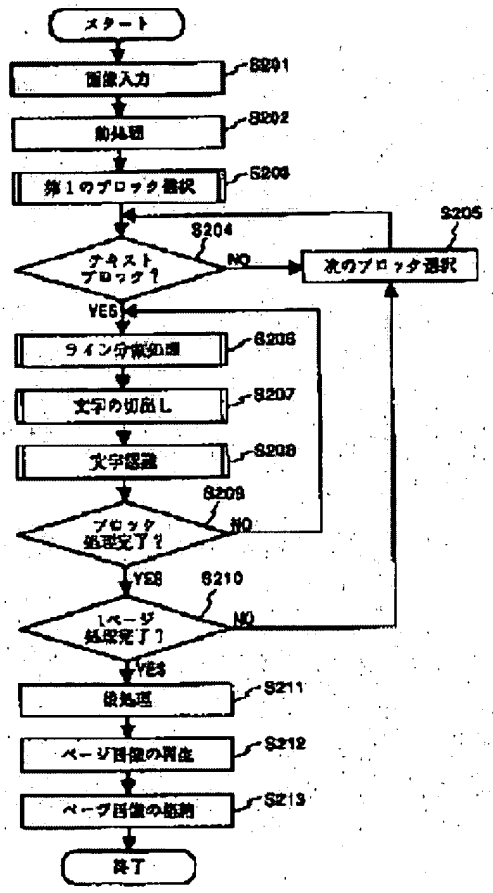
Also published as:

EP0567344 (A2)
US5680479 (A1)
EP0567344 (A3)
EP0567344 (B1)

Abstract of **JP6068301**

PURPOSE: To provide a character recognition method and device for recognizing a character on a document with a high speed and exactness, and preparing a text file.

CONSTITUTION: Pixel picture data of pixel units are inputted to a device (S201), the picture quality of the picture data is improved by a filter in order to improve deteriorated characters or pictures (S202), the hierarchical tree structure of the pictures is prepared (S203) so that text information/graphics information/ line information/picture information of information in a block can be identified, and each part in the picture can be reproduced in a proper sequence like a post-processing (S212), and whether or not the selected block is a text block is checked (S204). Then, a line dividing processing is operated to the text block (S206), each character in the line is segmented from another character in the line (S207), whether or not the processing to the text block and the processing to one page is ended is checked (S209 and 210), and a correction processing from an overall point of view such as context check or spelling check is operated (S211).



Data supplied from the ***esp@cenet*** database - Worldwide

T S1/5/1

1/5/1
DIALOG(R)File 347:JAPIO
(c) 2004 JPO & JAPIO. All rts. reserv.

04424401
METHOD AND DEVICE FOR RECOGNIZING CHARACTER

PUB. NO.: 06-068301 [JP 6068301 A]
PUBLISHED: March 11, 1994 (19940311)
INVENTOR(s): SHIN YAN WANGU
MEZAADOU AARU BAEZUII
KURISUTOFUAA EE SHIERITSUKU
APPLICANT(s): CANON INC [000100] (A Japanese Company or Corporation), JP (Japan)
CANON INF SYST INC [000000] (A Non-Japanese Company or Corporation), US (United States of America)
APPL. NO.: 05-121883 [JP 93121883]
FILED: April 26, 1993 (19930426)
PRIORITY: 7-873,012 [US 873012-1992], US (United States of America), April 24, 1992 (19920424)
INTL CLASS: [5] G06K-009/20; G06K-009/32; G06K-009/34
JAPIO CLASS: 45.3 (INFORMATION PROCESSING -- Input Output Units); 44.7 (COMMUNICATION -- Facsimile)
JAPIO KEYWORD:R002 (LASERS); R107 (INFORMATION PROCESSING -- OCR & OMR Optical Readers); R108 (INFORMATION PROCESSING -- Speech Recognition & Synthesis); R131 (INFORMATION PROCESSING -- Microcomputers & Microprocessers); R139 (INFORMATION PROCESSING -- Word Processors)
?

(19)日本国特許庁（ＪＰ）　(12) 公開特許公報（A）　(11)特許出願公開番号

特開平6－68301

(43)公開日　平成６年(1994)３月11日

(51)Int.Cl.[5]	識別記号	庁内整理番号	Ｆ Ｉ	技術表示箇所
Ｇ０６Ｋ　9/20	３４０　Ｌ			
9/32				
9/34				

審査請求　未請求　請求項の数286(全 420 頁)

(21)出願番号　特願平5－121883

(22)出願日　平成５年(1993)４月26日

(31)優先権主張番号　８７３０１２

(32)優先日　1992年４月24日

(33)優先権主張国　米国（US）

(71)出願人　000001007
キヤノン株式会社
東京都大田区下丸子３丁目30番２号

(71)出願人　592208172
キヤノン　インフォメーション　システムズ　インク.
Ｃａｎｏｎ　Ｉｎｆｏｒｍａｔｉｏｎ　Ｓｙｓｔｅｍｓ，　Ｉｎｃ.
アメリカ合衆国　カリフォルニア州　92626，コスタ　メサ，　ブルマン　ストリート　3188

(74)代理人　弁理士　大塚　康徳　(外１名)

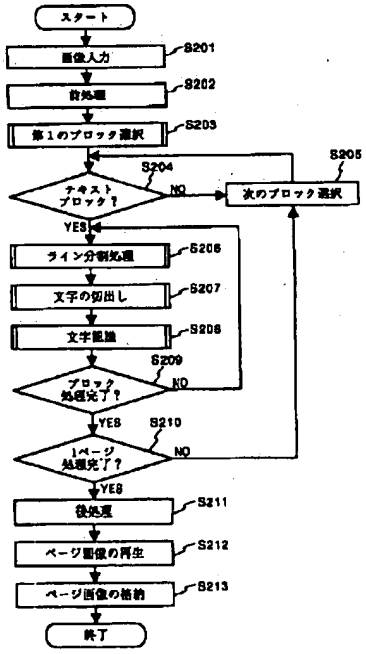
最終頁に続く

(54)【発明の名称】　文字認識方法及び装置

(57)【要約】　(修正有)

【目的】高速かつ正確に文書上の文字を認識してテキストファイルを作成する文字認識方法と装置の提供。

【構成】画素単位の画素画像データを装置に入力し（Ｓ２０１）、フイルターで画像データの画質を向上して劣化した文字や画像を改善し（Ｓ２０２）、ブロック中情報のテキスト情報／グラフイックス情報／線画情報／画像情報などの識別と画像中の各部分を後処理（Ｓ２１２）のような適切な順序で再生できるように画像の階層的木構造を生成し（Ｓ２０３）、選択されたブロックがテキストブロックか否かを調べ（Ｓ２０４）、テキストブロックに対してライン分割処理を行い（Ｓ２０６）、ライン中の各文字をライン中の他の文字から切り出し（Ｓ２０７）、テキストブロックに対する処理と１ページ分に対する処理が終了したかを調べ（Ｓ２０９，２１０）、文脈チェックやスペルチェックなどの全体的な見地からの修正処理を行う（Ｓ２１１）。



1

【特許請求の範囲】

【請求項１】　画素画像データから画素ブロックを選別する方法であって、

画素データ内で連結要素の輪郭を追跡する行程と、

輪郭が追跡された各連結要素を囲んで矩形を形成する行程と、

サイズと他の矩形への近接関係とに基づいて矩形を選択的に横方向に連結して、ラインを形成する第１の連結行程と、

サイズと他のラインへの近接関係とに基づいてラインを選択的に縦方向に連結して、ブロックを形成する第２の連結行程とを備える方法。

【請求項２】　更に、画素画像データを入力する行程を備え、

該入力行程は、画素画像データが２値画素画像データでない場合に、画素画像データを２値画素画像データに変換する行程を含む請求項１記載の方法。

【請求項３】　前記矩形形成行程では、各連結要素を囲む最小の矩形が形成される請求項１記載の方法。

【請求項４】　更に、前記矩形形成行程で形成された対応する矩形の位置に基づいて、前記輪郭追跡行程で追跡された連結要素の階層的木構造を形成する行程を備える請求項１記載の方法。

【請求項５】　更に、前記矩形形成行程で形成された矩形をテキスト要素と非テキスト要素とに分類する行程を備える請求項４記載の方法。

【請求項６】　更に、テキスト要素のブロック内の文字画像を認識する行程を備える請求項５記載の方法。

【請求項７】　前記分類行程は、要素の高さの所定のしきい値に対応して実行される請求項５記載の方法。

【請求項８】　前記分類行程は、前記矩形形成行程で形成された矩形の高さに関連する統計情報に対応して実行される請求項６記載の方法。

【請求項９】　前記第１の連結行程と第２の連結行程とは、非テキスト要素に対しては実行されない請求項６記載の方法。

【請求項１０】　更に、非テキスト要素に対し白輪郭を得る行程を備える請求項６記載の方法。

【請求項１１】　非テキスト要素は白輪郭の数に対応して表要素と見なされる請求項１０記載の方法。

【請求項１２】　更に、白輪郭の充填率を算出する行程を備える請求項１０記載の方法。

【請求項１３】　非テキスト要素は、充填率が高い場合には画像データと見なされない請求項１２記載の方法。

【請求項１４】　更に、非格子状に配置された白輪郭を再接続する行程を備える請求項１２記載の方法。

【請求項１５】　非テキスト要素は、再接続率が高くない場合に表と見なされる請求項１４記載の方法。

【請求項１６】　白輪郭は４方向で算出される請求項１０記載の方法。

2

【請求項１７】　連結要素の輪郭は、少なくとも８方向に追跡される請求項１記載の方法。

【請求項１８】　前記輪郭追跡行程では、連結要素の外部のみで連結要素の輪郭が追跡される請求項１記載の方法。

【請求項１９】　更に、画素画像データ内でギャップを検出する行程を備え、前記第１の連結行程では、ギャップが矩形を隔てている場合には矩形がラインに連結されない請求項１記載の方法。

【請求項２０】　矩形の間を縦に延びるギャップに対応して段を検出する請求項１９記載の方法。

【請求項２１】　前記第２の連結行程は、前記第１の連結行程で連結されたテキストデータのライン間に非テキストの境界を決定する行程を含み、前記第２の連結行程は、間に非テキストの境界がある場合にラインを縦方向にブロックとして連結しない請求項１記載の方法。

【請求項２２】　更に、前記輪郭追跡行程の前に画素画像データを圧縮する行程を備える請求項１記載の方法。

【請求項２３】　画素画像データ内の文字を認識する方法であって、

画素画像データ内で連結要素の輪郭を追跡する行程と、

追跡された連結要素がテキスト要素を含むか非テキスト要素を含むかを決める行程と、テキスト要素を横方向に選択的に連結してテキストラインを形成する行程と、テキストラインを選択的に縦方向に連結してテキストブロックを形成する行程とを含み、画素画像データから画素ブロックを選別する行程と、

テキストブロックを画素画像データのラインに分割する行程と、

前記分割行程で分割されたラインから文字を切り出す行程と、

前記切出行程で切り出された文字を認識する行程と、

前記認識行程で認識された文字を、前記ブロック選別行程で確定された順に対応して格納する行程とを備える方法。

【請求項２４】　更に、テキスト要素のブロック内の文字画像を認識する行程を備える請求項２３記載の方法。

【請求項２５】　更に、画素画像データが前処理される前処理行程を備える請求項２３記載の方法。

【請求項２６】　前記前処理行程は画像圧縮行程を含む請求項２５記載の方法。

【請求項２７】　前記前処理行程は画素画像データの画質を向上するフィルタ行程を含む請求項２５記載の方法。

【請求項２８】　更に、前記認識行程で認識された文字を後処理する行程を備える請求項２３記載の方法。

【請求項２９】　前記後処理は文脈チェックを含む請求項２８記載の方法。

【請求項３０】　更に、画素画像データを入力する行程を備え、

該入力行程は、画素画像データが２値画素画像データでない場合に、画素画像データを２値画素画像データに変換する行程を含む請求項２３記載の方法。

【請求項３１】　前記ブロック選別行程は、前記輪郭追跡行程で定められた連結画素に基づいて階層的木構造を形成する行程を含む請求項２３記載の方法。

【請求項３２】　更に、前記輪郭追跡行程で追跡された連結要素を囲む矩形を形成する行程を備える請求項３１記載の方法。

【請求項３３】　前記階層的木構造は、前記矩形形成行程で形成された矩形の位置に基づいて形成される請求項３２記載の方法。

【請求項３４】　更に、第１及び第２の連結行程に基づいて、階層的木構造を更新する行程を備える請求項３１記載の方法。

【請求項３５】　更に、連結要素をテキスト要素又は非テキスト要素に分類する行程を備える請求項３１記載の方法。

【請求項３６】　前記第１及び第２の連結行程は、テキスト要素を連結するが非テキスト要素は連結しない請求項３５記載の方法。

【請求項３７】　更に、非テキスト要素の内部で白輪郭を追跡する行程を備える請求項３５記載の方法。

【請求項３８】　更に、非テキスト要素に含まれる白輪郭の数に基づいて、非テキスト要素を表と見なす行程を備える請求項３７記載の方法。

【請求項３９】　更に、非格子状に配置された白輪郭を再接続する行程を備える請求項３７記載の方法。

【請求項４０】　前記非テキスト要素は、前記再接続行程での再接続に基づいて表と見なされる請求項３９記載の方法。

【請求項４１】　更に、非テキスト要素内の白輪郭の充填率を算出する行程を備える請求項３５記載の方法。

【請求項４２】　非テキスト要素は、充填率が高い場合に表と見なされない請求項４１記載の方法。

【請求項４３】　画素画像データ内で文字のテキストファイルを形成する方法であって、

画素画像データが２値の画素画像データでない場合に、画素画像データを２値の画素画像データに変換する行程を含む画素画像データの入力行程と、

画素画像データ内で連結要素の輪郭を追跡する行程と、

追跡された連結要素のサイズに基づいて連結要素がテキスト要素か非テキスト要素かを決める行程と、隣接するテキスト要素の近接関係に基づいてテキスト要素を選択的に横方向に連結してテキストラインを形成する行程と、隣接するテキストラインの近接関係とテキスト要素間の非テキスト要素の位置とに基づいてテキストラインを選択的に縦方向に連結してテキストブロックを形成する行程とを含み、画素画像データのブロックを選別する行程と、

前記輪郭追跡で追跡された画素要素に基づいて、階層的木構造を形成する行程と、

テキストブロックを画素画像データのラインに分割する行程と、

前記分割行程で分割されたラインから文字を切り出す行程と、

前記切出行程で切り出された文字を認識し、前記認識に基づいて文字コードを割り当てる行程と、

前記階層的木構造で確定された順に対応して文字コードをコンピュータのテキストファイルに格納する行程とを備える方法。

【請求項４４】　更に、画素画像データが前処理される前処理行程を備える請求項４３記載の方法。

【請求項４５】　前記前処理行程は画像圧縮行程を含む請求項４４記載の方法。

【請求項４６】　前記前処理行程は画素画像データの画質を向上する行程を含む請求項４５記載の方法。

【請求項４７】　更に、文脈チェックを含み、前記認識行程で認識された文字を後処理する行程を備える請求項４３記載の方法。

【請求項４８】　更に、非テキスト要素の内部で白輪郭を追跡する行程を備える請求項４３記載の方法。

【請求項４９】　更に、非テキストに識別子を付与する行程を更に備える請求項４８記載の方法。

【請求項５０】　更に、非テキスト要素に含まれる白輪郭の数に基づいて表識別子を付与する行程を備える請求項４９記載の方法。

【請求項５１】　更に、非格子状配置の白輪郭を再接続する行程を備え、非テキスト要素には前記再接続行程での再接続率に基づいて表の識別子が付与される請求項４９記載の方法。

【請求項５２】　更に、非テキスト要素内の白輪郭の充填率を算出する行程を備え、充填率が高くない場合に非テキスト要素に表の識別子を付与する請求項４９記載の方法。

【請求項５３】　画素画像データから画素ブロックを選別する装置であって、

画素データ内で連結要素の輪郭を追跡する輪郭追跡手段と、

前記輪郭追跡手段によって追跡された各連結要素を囲んで矩形を形成する矩形形成手段と、

サイズと他の矩形への近接関係とに基づいて矩形を選択的に横方向に連結して、ラインを形成する第１連結手段と、

サイズと他のラインへの近接関係に基づいてラインを選択的に縦方向に連結して、ブロックを形成する第２連結手段とを備える装置。

【請求項５４】　更に、画素画像データを入力する入力手段を備え、

該入力手段は、画素画像データが２値画素画像データで

5

ない場合に、画素画像データを２値画素画像データに変換する変換手段を含む請求項５３記載の装置。

【請求項５５】　前記矩形形成手段では、各連結要素を囲む最小の矩形が形成される請求項５３記載の装置。

【請求項５６】　更に、前記矩形形成手段で形成された対応する矩形の位置に基づいて、前記輪郭追跡手段で追跡された連結要素の階層的木構造を形成する第２の形成手段を備える請求項５３記載の装置。

【請求項５７】　更に、前記矩形形成手段で形成された矩形をテキスト要素と非テキスト要素とに分類する分類手段を備える請求項５６記載の装置。

【請求項５８】　更に、テキスト要素のブロック内の文字画像を認識する認識手段を備える請求項５７記載の装置。

【請求項５９】　前記分類手段は、要素の高さの所定のしきい値に対応して分類する請求項５７記載の装置。

【請求項６０】　前記分類手段は、前記矩形形成手段で形成された矩形の高さに関連する統計情報に対応して分類する請求項５７記載の装置。

【請求項６１】　前記第１連結手段と第２連結手段とは、非テキスト要素には使用されない請求項５７記載の装置。

【請求項６２】　更に、非テキスト要素に対して白輪郭を得る手段を備える請求項５７記載の装置。

【請求項６３】　非テキスト要素は白輪郭の数に対応して表の要素と見なされる請求項６２記載の装置。

【請求項６４】　更に、白輪郭の充填率を算出する算出手段を備える請求項６２記載の装置。

【請求項６５】　非テキスト要素は、充填率が高い場合には画像データと見なされない請求項６４記載の装置。

【請求項６６】　更に、非格子状に配置された白輪郭を再接続する再接続手段を備える請求項６４記載の装置。

【請求項６７】　非テキスト要素は、再接続率が高くない場合に表と見なされる請求項６６記載の装置。

【請求項６８】　白輪郭は４方向で算出される請求項６２記載の装置。

【請求項６９】　連結要素の輪郭は、少なくとも８方向に追跡される請求項５３記載の装置。

【請求項７０】　前記輪郭追跡手段では、連結要素の外部のみで連結要素の輪郭が追跡される請求項５３記載の装置。

【請求項７１】　更に、画素画像データ内でギャップを検出する検出手段を備え、前記第１連結手段では、ギャップが矩形を隔てている場合には矩形がラインに連結されない請求項５３記載の装置。

【請求項７２】　矩形の間を縦に延びるギャップに対応して段が検出される請求項７１記載の装置。

【請求項７３】　前記第２連結手段は、前記第１連結手段で連結されたテキストデータのライン間に非テキストの境界を決定する決定手段を含み、前記第２連結手段

6

は、間に非テキストの境界がある場合にラインを縦方向にブロックとして連結しない請求項５３記載の装置。

【請求項７４】　更に、前記輪郭追跡手段の前に、画素画像データを圧縮する圧縮手段を備える請求項５３記載の装置。

【請求項７５】　画素画像データ内の文字を認識する装置であって、

画素画像データ内で連結要素の輪郭を追跡する行程と、定められた連結要素がテキスト要素を含むか非テキスト要素を含むかを決める行程と、テキスト要素を横方向に選択的に連結してテキストラインを形成する行程と、テキストラインを選択的に縦方向に連結してテキストブロックを形成する行程とを含み、画素画像データから画素ブロックを選別するブロック選別手段と、

テキストブロックを画素画像データのラインに分割する分割手段と、

前記分割手段で分割されたラインから文字を切り出す切出手段と、

前記切出手段で切り出された文字を認識する認識手段と、

前記認識手段で認識された文字を、前記ブロック選別手段で確定された順に対応して格納する格納手段とを備える装置。

【請求項７６】　更に、テキスト要素のブロック内の文字画像を認識する第２の認識手段を備える請求項７５記載の装置。

【請求項７７】　更に、画素画像データが前処理される前処理手段を備える請求項７５記載の装置。

【請求項７８】　前記前処理手段は画像圧縮手段を含む請求項７７記載の装置。

【請求項７９】　前記前処理手段は画素画像データの画質を向上するフィルタ手段を含む請求項７７記載の装置。

【請求項８０】　更に、前記認識手段で認識された文字を後処理する後処理手段を備える請求項７５記載の装置。

【請求項８１】　前記後処理手段は文脈チェック手段を含む請求項８０記載の装置。

【請求項８２】　更に、画素画像データを入力する入力手段を備え、

該入力手段は、画素画像データが２値画素画像データでない場合に、画素画像データを２値画素画像データに変換する変換手段を含む請求項７５記載の装置。

【請求項８３】　前記ブロック選別手段は、前記輪郭追跡手段で追跡された連結要素に基づいて階層的木構造を形成する手段を含む請求項７５記載の装置。

【請求項８４】　更に、前記輪郭追跡手段で定められた連結要素を囲む矩形を形成する形成手段を備える請求項８３記載の装置。

【請求項８５】　前記階層的木構造は、前記矩形形成手

段で形成された矩形の位置に基づいて形成される請求項８４記載の装置。

【請求項８６】　更に、第１及び第２連結手段に基づいて、階層的木構造を更新する更新手段を備える請求項６４記載の装置。

【請求項８７】　更に、連結要素をテキスト要素又は非テキスト要素に分類する分類手段を備える請求項８３記載の装置。

【請求項８８】　前記第１及び第２連結手段は、テキスト要素を連結するが非テキスト要素は連結しない請求項８７記載の装置。

【請求項８９】　更に、非テキスト要素の内部で白輪郭を追跡する手段を備える請求項８７記載の装置。

【請求項９０】　更に、非テキスト要素を含まれる白輪郭の数に基づいて表と見なす手段を備える請求項８９記載の装置。

【請求項９１】　更に、非格子状に配置された白輪郭を再接続する再接続手段を備える請求項８９記載の装置。

【請求項９２】　前記非テキスト要素は、前記再接続手段での再接続に基づいて表と見なされる請求項９１記載の装置。

【請求項９３】　更に、非テキスト要素内の白輪郭の充填率を算出する算出手段を備える請求項８７記載の装置。

【請求項９４】　非テキスト要素は、充填率が高い場合に表と見なされない請求項９３記載の装置。

【請求項９５】　画素画像データ内で文字のテキストファイルを形成する装置であって、

画素画像データが２値の画素画像データでない場合に、画素画像データを２値の画素画像データに変換する変換手段を含む画素画像データの入力手段と、

画素画像データ内で連結要素の輪郭を追跡する輪郭追跡手段と、取り出された連結要素のサイズに基づいて連結要素がテキスト要素か非テキスト要素かを決める決定手段と、隣接するテキスト要素の近接関係に基づいてテキスト要素を選択的に横方向に連結してテキストラインを形成する第１連結手段と、隣接するテキストラインの近接関係とテキスト要素間の非テキスト要素の位置とに基づいてテキストラインを選択的に縦方向に連結してテキストブロックを形成する第２連結手段とを含み、画素画像データのブロックを選別するブロック選別手段と、

前記輪郭追跡で追跡された画素要素に基づいて、階層的木構造を形成する形成手段と、

テキストブロックを画素画像データのラインに分割する分割手段と、

前記分割手段で分割されたラインから文字を切り出す切出手段と、

前記切出手段で切り出された文字を認識し、前記認識に基づいて文字コードを割り当てる認識手段と、

前記階層的木構造で確定された順に対応して文字コードをコンピュータのテキストファイルに格納する格納手段とを備える装置。

【請求項９６】　更に、画素画像データが前処理される前処理手段を備える請求項９５記載の装置。

【請求項９７】　前記前処理手段は画像圧縮手段を含む請求項９６記載の装置。

【請求項９８】　前記前処理手段は画素画像データの画質向上手段を含む請求項９７記載の装置。

【請求項９９】　更に、文脈チェック手段を含み、前記認識手段で認識された文字を後処理する後処理手段を備える請求項９５記載の装置。

【請求項１００】　更に、非テキスト要素の内部で白輪郭を追跡する手段を備える請求項９５記載の装置。

【請求項１０１】　更に、非テキストに識別子を付与する付与手段を更に備える請求項１００記載の装置。

【請求項１０２】　更に、非テキスト要素に含まれる白輪郭の数に基づいて、非テキスト要素に表の識別子が付与される請求項１０１記載の装置。

【請求項１０３】　更に、非格子状配置の白輪郭を再接続する再接続手段を備え、非テキスト要素には前記再接続手段での再接続率に基づいて表の識別子が付与される請求項１０１記載の装置。

【請求項１０４】　更に、非テキスト要素内の白輪郭の充填率を算出する算出手段を備え、充填率が高くない場合に非テキスト要素に表識別子を付与する請求項１０１記載の装置。

【請求項１０５】　画素画像データでラインの位置を割り出す方法であって、

画像データの横方向に伸びた部分の全域で画像密度の水平投影を得る行程と、

前記水平投影に基づいて、画像密度が投影された部分の幅を減少するか否かを決定する行程と、

前記水平投影に基づいて、ラインの位置を割り出す行程を備える方法。

【請求項１０６】　前記決定行程は、実質的に空白の部分で水平投影を調べる行程と、実質的に空白の部分が見付からない場合に幅を減少する行程とを備える請求項１０５記載の方法。

【請求項１０７】　前記調べる行程は、間隔の小さい非空白の部分を連結する行程を含む請求項１０６記載の方法。

【請求項１０８】　前記水平投影を得る行程と決定する行程とは、順に繰り返し実行される請求項１０５記載の方法。

【請求項１０９】　前記割出行程では、水平投影の実質的に空白の領域間の画像データのライン位置を、ラインとして割り出す請求項１０５記載の方法。

【請求項１１０】　更に、画素画像データを前処理する前処理行程を備える請求項１０５記載の方法。

【請求項１１１】　段に分割可能な画素画像データ内で

9

ラインの位置を割り出す方法であって、
画像データの少なくとも1つの横方向に伸びる段の全域で、画像密度の水平投影を得る行程と、
前記行程で得られた水平投影に基づいて、画像密度が投影される段の数を増加するか否かを決定する行程と、
前記水平投影に基づいてデジタル画像データ内でラインの位置を割り出す行程とを備える方法。

【請求項112】　前記決定行程は、実質的に空白の部分で水平投影を調べる行程と、実質的に空白の部分が見付からない場合に、段の数を増加する行程とを備える請求項111記載の方法。

【請求項113】　前記調べる行程は間隔の小さい非空白の部分を連結する行程を含む請求項112記載の方法。

【請求項114】　前記水平投影を得る行程と決定する行程とは、順に繰り返し実行される請求項111記載の方法。

【請求項115】　前記割出行程では、水平投影の実質的に空白の領域間の画像データのライン位置を、ラインとして割り出す請求項111記載の方法。

【請求項116】　更に、画像データの隣接する段のラインが重なるか否かを決める行程を備える請求項111記載の方法。

【請求項117】　前記ラインが重なるかを決める行程は、スケール不変の決定から成る請求項116記載の方法。

【請求項118】　更に、画素画像データを前処理する前処理行程を備え、前記前処理行程は画像縮小行程を含む請求項111記載の方法。

【請求項119】　前記画像縮小は水平方向と垂直方向とで異なる請求項118記載の方法。

【請求項120】　画素画像データのラインセグメントから文字画像を切り出す方法であって、
ラインセグメントから非接触で重なっていない文字を切り出す第1の切出行程と、
ラインセグメントから接触文字を切り出す第2の切出行程とを備える方法。

【請求項121】　更に、高解像度の画像セグメントを低解像度に圧縮する画像データのイメージセグメントの前処理行程を備える請求項120記載の方法。

【請求項122】　垂直方向と水平方向とで異なる圧縮率が使われる請求項121記載の方法。

【請求項123】　更に、前記第1の切出行程で切り出された文字を認識処理する行程を備え、前記第2の切出行程は前記認識行程で認識されなかった文字に対してのみ実行される請求項120記載の方法。

【請求項124】　更に、非接触で重なっている文字データをラインセグメントから切り出す中間の切出行程を備える請求項123記載の方法。

【請求項125】　前記中間切出行程は、非接触で重なっている文字を画素データの輪郭の追跡により切り出す行程を含む請求項124記載の方法。

10

【請求項126】　前記中間切出行程は、間違って切り出された文字を再接続する行程を含む請求項124記載の方法。

【請求項127】　更に、前記認識行程で認識されなかった文字を再接続する行程を備える請求項123記載の方法。

【請求項128】　前記第1の切出行程は、前記ラインセグメントを非空白の画素データが見付かるまでとびとびに処理する行程を含む請求項120記載の方法。

【請求項129】　更に、前記とびとびの処理行程で非空白の画素が見付かった場合に、緻密に前後をサーチする行程を備える請求項128記載の方法。

【請求項130】　前記第2の切出行程は非垂直切り出しによって接触文字を切り出す行程を含む請求項120記載の方法。

【請求項131】　非垂直切り出しの角度は画素密度の垂直投影特性に対応して決められる請求項130記載の方法。

【請求項132】　更に、垂直投影特性に基づいて少なくとも1つの回転された投影特性を得る行程を備えることを特徴とする131記載の方法。

【請求項133】　更に、前記垂直投影特性の角度に近接する角度で複数の回転された投影特性を得る行程を備える請求項132記載の方法。

【請求項134】　非垂直切り出しの角度は、前記回転投影特性を得る行程と垂直投影特性を得る行程とから得られた投影特性の最小値に基づいて得られる請求項133記載の方法。

【請求項135】　前記第2の切出行程は、前記ラインセグメント内の文字の特徴が既知かどうかに基づいて選択可能である請求項120記載の方法。

【請求項136】　前記第2の切出行程は、前記文字セット内の文字データのスペースの統計に基づいて切り出す行程を含む請求項135記載の方法。

【請求項137】　更に、小さな文字のセットを再接続する行程を備える請求項136記載の方法。

【請求項138】　文字画素データのラインセグメント内で接触文字を切り出す方法であって、
ラインセグメント内で画素データの垂直投影特性を得る行程と、
垂直投影特性の最小値から隣接する最大値への方向から少なくとも1つの角度を算出する第1の算出行程と、
前記第1の算出行程で算出された角度に基づいて、少なくとも1つの回転された投影特性を算出する第2の算出行程と、
前記回転投影特性の最小値に対応する位置で前記回転投影特性の角度にラインセグメントを切る行程とを備える方法。

11

【請求項１３９】　前記第１の算出行程は、前記垂直投影特性を複数のしきい値と比較して前記最小値と隣接する最大値との位置を決定する行程を含む請求項１３８記載の方法。

【請求項１４０】　更に、最小値と隣接する最大値とが見付からない場合に、前記しきい値を増加させる行程を備える請求項１３９記載の方法。

【請求項１４１】　最小値は、少なくとも一方の側が第２しきい値より大きい最大値によって囲まれ、第１しきい値より小さな垂直投影特性上の位置として割り出される請求項１３９記載の方法。

【請求項１４２】　更に、各々が前記第２の算出行程で算出された回転投影特性に近接する角度で算出される複数の回転投影特性を算出する行程を備える請求項１３８記載の方法。

【請求項１４３】　更に、複数の回転投影特性と垂直投影特性との全体の最小値を割り出す行程を備え、前記切断行程は最小値の位置に対応する位置で最小値が得られた角度に対応する角度に切る請求項１４２記載の方法。

【請求項１４４】　画素画像データ内で文字を認識する方法であって、

画素画像データから画素ブロックを選別する行程と、

少なくとも１つの段全体の画素密度の水平投影に基づいてテキストブロックを少なくとも１つの段に適応的に分配することにより、テキストブロックを画素データのラインに分割する行程と、

非接触で重なっていない文字がラインセグメントから切り出される第１の切出行程と、接触文字を切り出す少なくとも１つの追加の切出行程とを含み、前記分割行程で分割されたラインセグメントから文字を切り出す行程と、

前記文字切出行程で切り出された文字を認識する行程と、

前記認識行程で認識された文字を前記ブロック選別行程で確定された順に対応して格納する行程とを備える方法。

【請求項１４５】　更に、画素画像データが２値の画素画像データでない場合に、画素画像データを２値の画素画像データに変換する行程を含む画素画像データを入力する行程を備える請求項１４４記載の方法。

【請求項１４６】　更に、画素画像データが前処理される前処理行程を備える請求項１４４記載の方法。

【請求項１４７】　前記前処理行程は、画像圧縮行程を含む請求項１４６記載の方法。

【請求項１４８】　前記前処理行程は、画素画像データの画質を向上するフィルタ行程を含む請求項１４６記載の方法。

【請求項１４９】　更に、前記認識行程で認識された文字を後処理する行程を備える請求項１４４記載の方法。

【請求項１５０】　前記後処理は文脈チェックを含む請求項１４９記載の方法。

12

【請求項１５１】　前記第１の切出行程はラインセグメントをとびとびに処理する行程を含む請求項１４４記載の方法。

【請求項１５２】　前記追加の切出行程は、画素画像データの文字について特徴が既知か否かに対応して選択可能である請求項１４４記載の方法。

【請求項１５３】　前記追加の切出行程では既知の文字セットの統計に基づいて切り出しを行う請求項１５２記載の方法。

【請求項１５４】　前記追加の切出行程では画素密度の回転投影に対応して切り出しを行う請求項１５２記載の方法。

【請求項１５５】　画素密度の回転投影は画素密度の垂直投影に基づいて決められた角度で得られる請求項１５４記載の方法。

【請求項１５６】　更に、前記第１の切出行程と前記少なくとも１つの追加の切出行程の間に中間の切出行程を備える請求項１５４記載の方法。

【請求項１５７】　前記中間の切出行程はラインセグメント内の重なった画素画像の輪郭を追跡する行程を含む請求項１５６記載の方法。

【請求項１５８】　更に、文字が間違って切り出された場合に、前記中間の切出行程で切り出された文字を再接続する行程を備える請求項１５７記載の方法。

【請求項１５９】　前記認識行程は、前記追加の切出行程の前に前記第１の切出行程で切り出された文字を認識し、前記追加の切出行程は前記認識行程で認識されない文字に対して行なわれる請求項１５８記載の方法。

【請求項１６０】　更に、文字が前記認識行程で認識されない場合に、前記少なくとも１つの追加の切出行程で切り出された文字を再接続する行程を備える請求項１５９記載の方法。

【請求項１６１】　画素画像データから文字のテキストファイルを形成する方法であって、

画素画像データが２値の画素画像データでない場合に、画素画像データを２値の画素画像データに変換する行程を含む画素画像データの入力行程と、

画素画像データのブロックがテキストブロックか非テキストブロックかに対応して画素画像データのブロックを選別する行程と、

テキスト及び非テキストブロックの階層的木構造を得る行程と、

少なくとも１つの段の全域の画素密度の水平投影に基づいて、テキストブロックを前記少なくとも１つの段で適応的に分離することにより、テキストブロックを画素画像データのラインに分割する行程と、

非接触で重なっていない文字をラインセグメントから切り出す第１の切出行程と、接触文字を切り出す少なくとも１つの追加の切出行程とを含み、前記分割行程で分割

されたラインから文字を切り出す行程とを備える方法。

【請求項１６２】　更に、画素画像データが前処理される前処理行程を備える請求項１６１記載の方法。

【請求項１６３】　前記前処理行程は画像圧縮行程を含む請求項１６２記載の方法。

【請求項１６４】　前記前処理行程は画素画像データの画質を向上する行程を含む請求項１６３記載の方法。

【請求項１６５】　更に、文脈チェックを含み、前記認識行程で認識された文字を後処理する行程を備える請求項１６１記載の方法。

【請求項１６６】　前記第１の切出行程はラインセグメントをとびとびに処理する行程を含む請求項１６１記載の方法。

【請求項１６７】　前記追加の切出行程は、画素画像データの文字について特徴が既知か否かに対応して選択可能である請求項１６６記載の方法。

【請求項１６８】　前記追加の切出行程では既知の文字セットの統計に基づいて切り出しを行う請求項１６７記載の方法。

【請求項１６９】　前記追加の切出行程では画素密度の回転投影に対応して切り出しを行う請求項１６７記載の方法。

【請求項１７０】　画素密度の回転投影は画素密度の垂直投影に基づいて決められた角度で得られる請求項１６９記載の方法。

【請求項１７１】　更に、前記第１の切出行程と前記少なくとも１つの追加の切出行程の間に中間の切出行程を備える請求項１６１記載の方法。

【請求項１７２】　前記中間の切出行程はラインセグメント内の重なった画素画像の輪郭を追跡する行程を含む請求項１７１記載の方法。

【請求項１７３】　更に、文字が間違って切り出された場合に、前記中間の切出行程で切り出された文字を再接続する行程を備える請求項１７２記載の方法。

【請求項１７４】　前記認識行程は、前記追加の切出行程の前に前記第１の切出行程で切り出された文字を認識し、前記追加の切出行程は前記認識行程で認識されない文字に対して行なわれる請求項１７３載の方法。

【請求項１７５】　更に、文字が前記認識行程で認識されない場合に、前記少なくとも１つの追加の切出行程で切り出された文字を再接続する行程を備える請求項１７４記載の方法。

【請求項１７６】　画素画像データから文字のテキストファイルを形成する方法であって、

画素画像データが２値の画素画像データでない場合に、画素画像データを２値の画素画像データに変換する行程を含み、画素画像データを入力する行程と、

画素画像データ内で連結要素の輪郭を追跡する行程と、

前記追跡行程の間に追跡された連結要素のサイズに基づいて追跡された連結要素がテキスト要素を含むか非テキスト要素を含むから決める行程と、隣接するテキスト要素の近接関係に基づいてテキスト要素を選択的に横方向に連結してテキストラインを形成する行程と、隣接するテキストラインの近接関係とテキストライン間の非テキスト要素の位置とに基づいてテキストラインを選択的に縦方向に連結する行程とを含み、画素画像データのブロックを選別する行程と、

前記輪郭追跡行程で追跡された連結要素に基づいて、階層的木構造を形成する行程と、

少なくとも１つの段の全域での画素密度の水平投影に基づいて、テキストブロックを前記少なくとも１つの段に適応的に分離することによって、テキストブロックを画素データのラインに分割する行程と、

非接触で重なっていない文字をラインセグメントから切り出す第１の切出行程と、接触文字を切り出す少なくとも１つの追加の切出行程とを含み、前記分割行程で分割されたラインから文字を切り出す行程と、

前記文字切出行程で切り出された文字を認識し、前記認識に基づいて文字コードを割り付ける行程と、

階層的木構造により確定された順に対応して、文字コードをコンピュータのテキストファイルに格納する行程とを備える方法。

【請求項１７７】　更に、画素画像データが前処理される前処理行程を備える請求項１７６記載の方法。

【請求項１７８】　前記前処理行程は画像圧縮行程を含む請求項１７７記載の方法。

【請求項１７９】　前記前処理行程は画素画像データの画質を向上する行程を含む請求項１７８記載の方法。

【請求項１８０】　更に、文脈チェックを含み、前記認識行程で認識された文字を後処理する行程を備える請求項１７６記載の方法。

【請求項１８１】　前記第１の切出行程はラインセグメントをとびとびに処理する行程を含む請求項１７６記載の方法。

【請求項１８２】　前記追加の切出行程は、画素画像データの文字について特徴が既知か否かに対応して選択可能である請求項１７６記載の方法。

【請求項１８３】　前記追加の切出行程では既知の文字セットの統計に基づいて切り出しを行う請求項１８２記載の方法。

【請求項１８４】　前記追加の切出行程では画素密度の回転投影に対応して切り出しを行う請求項１８２記載の方法。

【請求項１８５】　画素密度の回転投影は画素密度の垂直投影に基づいて決められた角度で得られる請求項１８４記載の方法。

【請求項１８６】　更に、前記第１の切出行程と前記少なくとも１つの追加の切出行程の間に中間の切出行程を備える請求項１７６記載の方法。

【請求項１８７】　前記中間の切出行程はラインセグメ

15

ント内の重なった画素画像の輪郭を追跡する行程を含む請求項１８６記載の方法。

【請求項１８８】　更に、文字が間違って切り出された場合に、前記中間の切出行程で切り出された文字を再接続する行程を備える請求項１８７記載の方法。

【請求項１８９】　前記認識行程は、前記追加の切出行程の前に前記第１の切出行程で切り出された文字を認識し、前記追加の切出行程は前記認識行程で認識されない文字に対して行なわれる請求項１８８載の方法。

【請求項１９０】　更に、文字が前記認識行程で認識されない場合に、前記少なくとも１つの追加の切出行程で切り出された文字を再接続する行程を備える請求項１８９記載の方法。

【請求項１９１】　更に、非テキスト要素の内部で白輪郭を追跡する行程を備える請求項１７６記載の方法。

【請求項１９２】　更に、非テキスト要素に識別子を付加する行程を備える請求項１９１記載の方法。

【請求項１９３】　非テキスト要素内に含まれる白輪郭の数に基づいて非テキスト要素を表と見なす請求項１９２記載の方法。

【請求項１９４】　更に、非格子状配置の白輪郭を再接続する行程を備え、非テキスト要素には前記再接続行程での再接続率に基づいて表識別子が付加される請求項１９２記載の方法。

【請求項１９５】　更に、非テキスト要素内の白輪郭の充填率を算出する行程を備え、充填率が高くない場合に非テキスト要素に表識別子を付加する請求項１９２記載の方法。

【請求項１９６】　画素画像データでラインの位置を割り出す装置であって、
画像データの横方向に伸びた部分の全域で画像密度の水平投影を得る手段と、
水平投影に基づいて、画像密度が投影された部分の幅を減少するか否かを決定する決定行程と、
水平投影に基づいて、ラインの位置を割り出す割出行程とを備える装置。

【請求項１９７】　前記決定手段は、実質的に空白の部分で水平投影を調べる手段と、実質的に空白の部分が見付からない場合に幅を減少する減少手段とを備える請求項１９６記載の装置。

【請求項１９８】　前記調べる手段は間隔の小さい非空白の部分を接続する接続手段を含む請求項１９７記載の装置。

【請求項１９９】　前記水平投影を得る手段と決定手段とは、順に繰り返し使用される請求項１９６記載の装置。

【請求項２００】　前記割出手段では、水平投影の実質的に空白の領域間の画像データのライン位置を、ラインとして割り出す請求項１９６記載の装置。

【請求項２０１】　更に、画素画像データを前処理する

16

前処理手段を備える請求項１９６記載の装置。

【請求項２０２】　段に分離可能な画素画像データ内でラインの位置を割り出す装置であって、
画像データの少なくとも１つの横方向に伸びる段の全域で、画像密度の水平投影を得る手段と、
前記手段で得られた水平投影に基づいて、画像密度が投影される段の数を増加するか否かを決定する決定手段と、
前記水平投影に基づいてデジタル画像データ内でラインの位置を割り出す割出手段とを備える装置。

【請求項２０３】　前記決定手段は、実質的に空白の部分で水平投影を調べる手段と、実質的に空白の部分が見付からない場合に段の数を増加する増加手段とを備える請求項２０２記載の装置。

【請求項２０４】　前記調べる手段は間隔の小さい非空白の部分を連結する連結手段を含む請求項２０３記載の装置。

【請求項２０５】　前記水平投影を得る手段と決定手段とは、順に繰り返し使用される請求項２０２記載の装置。

【請求項２０６】　前記割出手段では、水平投影の実質的に空白の領域間の画像データのライン位置を、ラインとして割り出す請求項２０２記載の装置。

【請求項２０７】　更に、画像データの隣接する段のラインが重なるか否かを決める第２の決定手段を備える請求項２０２記載の装置。

【請求項２０８】　前記第２の決定手段は、画像データの隣接する段のブロックが重なるか否かを決定するスケール不変の決定手段から成る請求項２０７記載の装置。

【請求項２０９】　更に、画素画像データを前処理する前処理手段を備え、前記前処理手段は画像縮小手段を含む請求項２０２記載の装置。

【請求項２１０】　前記画像縮小は水平方向と垂直方向とで異なる請求項２０９記載の装置。

【請求項２１１】　画素画像データのラインセグメントから文字画像を切り出す装置であって、
ラインセグメントから非接触で重なっていない文字を切り出す第１の切出手段と、
ラインセグメントから接触文字を切り出す第２の切出手段とを備える装置。

【請求項２１２】　更に、高解像度の画像セグメントを低解像度に圧縮するように画像データのイメージセグメントを前処理する前処理手段を備える請求項２１１記載の装置。

【請求項２１３】　垂直方向と水平方向とで異なる圧縮率が使われる請求項２１２記載の装置。

【請求項２１４】　更に、前記第１の切出手段で切り出された文字を認識処理する認識手段を備え、前記第２の切出手段は前記認識手段で認識されなかった文字に対してのみ使用される請求項２１２記載の装置。

【請求項２１５】　更に、非接触で重なった文字データをラインセグメントから中間で切り出す第３の切出手段を備える請求項２１４記載の装置。

【請求項２１６】　前記第３の切出手段は、非接触で重なった文字を画素データの輪郭を追跡する輪郭追跡手段を含む請求項２１５記載の装置。

【請求項２１７】　前記第３の切出手段は、間違って切り出された文字を再接続する再接続手段を含む請求項２１５記載の装置。

【請求項２１８】　更に、前記認識手段で認識されなかった文字を再接続する第２の再接続手段を備える請求項２１４記載の装置。

【請求項２１９】　前記第１の切出手段は、前記ラインセグメントを非空白の画素データが見付かるまでとびとびに処理する手段を含む請求項２１１記載の装置。

【請求項２２０】　更に、前記とびとびの処理手段で非空白の画素が見付かった場合に、緻密に前後をサーチする検索手段を備える請求項２１９記載の装置。

【請求項２２１】　前記第２の切出手段は非垂直切り出しによって接触文字を切り出す請求項２１３記載の装置。

【請求項２２２】　非垂直切り出しの角度は画素密度の垂直投影特性に対応して決められる請求項２２１記載の装置。

【請求項２２３】　更に、垂直投影特性に基づいて少なくとも１つの回転された投影特性を得る手段を備えることを特徴とする２２２記載の装置。

【請求項２２４】　更に、前記垂直投影特性の角度に近接する角度で複数の回転された投影特性を得る手段を備える請求項２２３記載の装置。

【請求項２２５】　非垂直切り出しの角度は、前記前記回転投影特性を得る手段と垂直投影特性を得る手段とから得られた投影特性の最小値に基づいて得られる請求項２２４記載の装置。

【請求項２２６】　前記第２の切出手段は、前記ラインセグメント内の文字の特徴が既知かどうかに基づいて選択可能である請求項２１１記載の装置。

【請求項２２７】　前記第２の切出手段は、前記文字セット内の文字データのスペースの統計に基づいて切り出す請求項２２６記載の装置。

【請求項２２８】　更に、小さな文字のセットを再接続する第３の再接続手段を備える請求項２２７記載の装置。

【請求項２２９】　文字画素データのラインセグメント内で接触文字を切り出す装置であって、
ラインセグメント内で画素データの垂直投影特性を得る手段と、
垂直投影特性の最小値から隣接する最大値への方向から少なくとも１つの角度を算出する第１の算出手段と、
前記第１の算出手段で算出された角度に基づいて、少なくとも１つの回転された投影特性を算出する第２の算出手段と、
前記回転投影特性の最小値に対応する位置で前記回転投影特性の角度にラインセグメントを切る切断手段とを備える装置。

【請求項２３０】　前記第１の算出手段は、前記垂直投影特性を複数のしきい値と比較して前記最小値と隣接する最大値との位置を決定する比較手段を含む請求項２２９記載の装置。

【請求項２３１】　更に、最小値と隣接する最大値とが見付からない場合に、前記しきい値を増加させる増加手段を備える請求項２３０記載の装置。

【請求項２３２】　最小値は、少なくとも一方の側が第２しきい値より大きい最大値によって囲まれ、第１しきい値より小さな垂直投影特性上の位置として割り出される請求項２３１記載の装置。

【請求項２３３】　更に、各々が前記第２の算出手段で算出された回転投影特性に近接する角度で算出される複数の回転投影特性を算出する第３の算出手段を備える請求項２２９記載の装置。

【請求項２３４】　更に、複数の回転投影特性と垂直投影特性との全体の最小値を割り出す割出手段を備え、前記切断手段は最小値の位置に対応する位置で最小値が得られた角度に対応する角度に切る請求項２３３記載の装置。

【請求項２３５】　画素画像データ内で文字を認識する装置であって、
画素画像データから画素ブロックを選別する選別手段と、
少なくとも１つの段全体の画素密度の水平投影に基づいてテキストブロックを少なくとも１つの段に適応的に分配することにより、テキストブロックを画素データのラインに分割する分割手段と、
非接触で重なっていない文字がラインセグメントから切り出される第１の切出手段と、接触文字を切り出す少なくとも１つの追加の切出手段とを含み、前記分割手段で分割されたラインセグメントから文字を切り出す切出手段と、
前記切出手段で切り出された文字を認識する認識手段と、
前記認識手段で認識された文字を前記選別手段で確定された順に対応して格納する格納手段とを備える装置。

【請求項２３６】　更に、画素画像データが２値の画素画像データでない場合に、画素画像データを２値の画素画像データに変換する変換手段を含む画素画像データの入力手段を備える請求項２３５記載の装置。

【請求項２３７】　更に、画素画像データが前処理される前処理手段を備える請求項２３５記載の装置。

【請求項２３８】　前記前処理手段は画像圧縮手段を含む請求項２３７記載の装置。

19

【請求項２３９】　前記前処理手段は画素画像データの画質を向上するフィルタ手段を含む請求項２３７記載の装置。

【請求項２４０】　更に、前記認識手段で認識された文字を後処理する後処理手段を備える請求項２３５記載の装置。

【請求項２４１】　前記後処理手段は文脈チェック手段を含む請求項２４０記載の装置。

【請求項２４２】　前記第１の切出手段はラインセグメントをとびとびに処理する手段を含む請求項２３５記載の装置。

【請求項２４３】　前記追加の切出手段は、画素画像データの文字について特徴が既知か否かに対応して選択可能である請求項２３５記載の装置。

【請求項２４４】　前記追加の切出手段は既知の文字セットの統計に基づいて切り出しを行う請求項２４３記載の装置。

【請求項２４５】　前記追加の切出手段は画素密度の回転投影に対応して切り出しを行う請求項２４３記載の装置。

【請求項２４６】　画素密度の回転投影は画素密度の垂直投影に基づいて決められた角度で得られる請求項２４５記載の装置。

【請求項２４７】　更に、前記第１の切出手段の使用後で前記少なくとも１つの追加の切出手段の使用前に、中間の階層を切り出す中間の切出手段を備える請求項２４５記載の装置。

【請求項２４８】　前記中間の切出手段はラインセグメント内の重なった画素画像の輪郭を追跡する輪郭追跡手段を含む請求項２４７記載の装置。

【請求項２４９】　更に、文字が間違って切り出された場合に、前記中間の切出手段で切り出された文字を再接続する再接続手段を備える請求項２４８記載の装置。

【請求項２５０】　前記認識手段は、前記追加の切出手段の前に前記第１の切出手段で切り出された文字を認識し、前記追加の切出手段は前記認識手段で認識されない文字に対して行なわれる請求項２４９記載の装置。

【請求項２５１】　更に、文字が前記認識手段で認識されない場合に、前記少なくとも１つの追加の切出手段で切り出された文字を再接続する第２の再接続手段を備える請求項２５０記載の装置。

【請求項２５２】　画素画像データから文字のテキストファイルを形成する装置であって、
画素画像データが２値の画素画像データでない場合に、画素画像データを２値の画素画像データに変換する変換手段を含む画素画像データの入力手段と、
画素画像データのブロックがテキストブロックか非テキストブロックかに対応して画素画像データのブロックを選別する選別手段と、
テキスト及び非テキストブロックの階層的木構造を得る手段と、
少なくとも１つの段の全域の画素密度の水平投影に基づいて、テキストブロックを前記少なくとも１つの段に適応的に分配することにより、テキストブロックを画素画像データのラインに分割する分割手段と、
非接触で重なっていない文字をラインセグメントから切り出す第１の切出手段と、接触文字を切り出す少なくとも１つの追加の切出手段とを含み、前記分割手段で分割されたラインから文字を切り出す切出手段と、
前記切出手段で切り出された文字を認識し、前記認識に基づいて文字コードを割り付ける認識手段と、
前記階層的木構造により確立された順に対応して文字コードが格納されるコンピュータのテキストファイルとを備える装置。

20

【請求項２５３】　更に、画素画像データが前処理される前処理手段を備える請求項２５２記載の装置。

【請求項２５４】　前記前処理手段は画像圧縮手段を含む請求項２５３記載の装置。

【請求項２５５】　前記前処理手段は画素画像データの画質を向上する画質向上手段を含む請求項２５３記載の装置。

【請求項２５６】　更に、文脈チェック手段を含み、前記認識手段で認識された文字を後処理する後処理手段を備える請求項２５２記載の装置。

【請求項２５７】　前記第１の切出手段はラインセグメントをとびとびに処理する手段を含む請求項２５２記載の装置。

【請求項２５８】　前記追加の切出手段は、画素画像データの文字について特徴が既知か否かに対応して選択可能である請求項２５２記載の装置。

【請求項２５９】　前記追加の切出手段は既知の文字セットの統計に基づいて切り出しを行う請求項２５８記載の装置。

【請求項２６０】　前記追加の切出手段は画素密度の回転投影に対応して切り出しを行う請求項２５８記載の装置。

【請求項２６１】　画素密度の回転投影は画素密度の垂直投影に基づいて決められた角度で得られる請求項２６０記載の装置。

【請求項２６２】　更に、前記第１の切出手段と前記少なくとも１つの追加の切出手段の間に中間の階層を切り出す中間の切出手段を備える請求項２５２記載の装置。

【請求項２６３】　前記中間の切出手段はラインセグメント内の重なった画素画像の輪郭を追跡する輪郭追跡手段を含む請求項２６２記載の装置。

【請求項２６４】　更に、文字が間違って切り出された場合に、前記中間の切出手段で切り出された文字を再接続する再接続手段を備える請求項２６３記載の装置。

【請求項２６５】　前記認識手段は、前記追加の切出手段の前に前記第１の切出手段で切り出された文字を認識

し、前記追加の切出手段は前記認識手段で認識されない文字に対して行なわれる請求項２６４載の装置。

【請求項２６６】　更に、文字が前記認識手段で認識されない場合に、前記少なくとも１つの追加の切出手段で切り出された文字を再接続する第２の再接続手段を備える請求項２６５記載の装置。

【請求項２６７】　画素画像データから文字のテキストファイルを形成する装置であって、

画素画像データが２値の画素画像データでない場合に、画素画像データを２値の画素画像データに変換する変換手段を含み、画素画像データを入力する入力手段と、

画素画像データ内で連結要素の輪郭を追跡する輪郭追跡手段と、追跡された連結要素のサイズに基づいて追跡された連結要素がテキスト要素を含むか非テキスト要素を含むから決める決定手段と、隣接するテキスト要素の近接関係に基づいてテキスト要素を選択的に横方向に連結してテキストラインを形成する第１の連結手段と、隣接するテキストラインの近接関係とテキストライン間の非テキスト要素の位置とに基づいてテキストラインを選択的に縦方向に連結する第２の連結手段とを含み、画素画像データのブロックを選別する選別手段と、

前記輪郭追跡手段で追跡された連結要素に基づいて、階層的木構造を形成する形成手段と、

少なくとも１つの段の全域での画素密度の水平投影に基づいて、テキストブロックを前記少なくとも１つの段に適応的に分配することによって、テキストブロックを画素データのラインに分割する分割手段と、

非接触で重ならない文字をラインセグメントから切り出す第１の切出手段と、接触文字を切り出す少なくとも１つの追加の切出手段とを含み、前記分割手段で分割されたラインから文字を切り出す切出手段と、

前記文字切出手段で切り出された文字を認識し、前記認識に基づいて文字コードを割り付ける認識手段と、

階層的木構造により確定された順に対応して文字コードをコンピュータのテキストファイルに格納する格納手段とを備える装置。

【請求項２６８】　更に、画素画像データが前処理される前処理手段を備える請求項２６７記載の装置。

【請求項２６９】　前記前処理手段は画像圧縮手段を含む請求項２６８記載の装置。

【請求項２７０】　前記前処理手段は画素画像データの画質を向上する画質向上手段を含む請求項２６９記載の装置。

【請求項２７１】　更に、文脈チェック手段を含み、前記認識手段で認識された文字を後処理する後処理手段を備える請求項２６７記載の装置。

【請求項２７２】　前記第１の切出手段はラインセグメントをとびとびに処理する手段を含む請求項２６７記載の装置。

【請求項２７３】　前記追加の切出手段は画素画像データの文字について特徴が既知か否かに対応して選択可能である請求項２６７記載の装置。

【請求項２７４】　前記追加の切出手段は既知の文字セットの統計に基づいて切り出しを行う請求項２７３記載の装置。

【請求項２７５】　前記追加の切出手段は画素密度の回転投影に対応して切り出しを行う請求項２７３記載の装置。

【請求項２７６】　画素密度の回転投影は画素密度の垂直投影に基づいて決められた角度で得られる請求項２７５記載の装置。

【請求項２７７】　更に、前記第１の切出手段の後で前記少なくとも１つの追加の切出手段の前に中間の階層を切り出す中間の切出手段を備える請求項２６７記載の装置。

【請求項２７８】　前記中間の切出手段はラインセグメント内の重なった画素画像の輪郭を追跡する輪郭追跡手段を含む請求項２７７記載の装置。

【請求項２７９】　更に、文字が間違って切り出された場合に、前記中間の切出手段で切り出された文字を再接続する第１の再接続手段を備える請求項２７８記載の装置。

【請求項２８０】　前記認識手段は、前記追加の切出手段の前に前記第１の切出手段で切り出された文字を認識し、前記追加の切出手段は前記認識手段で認識されない文字に対して行なわれる請求項２７９載の装置。

【請求項２８１】　更に、文字が前記認識手段で認識されない場合に、前記少なくとも１つの追加の切出手段で切り出された文字を再接続する第２の再接続手段を備える請求項２８０記載の装置。

【請求項２８２】　更に、非テキスト要素の内部で白輪郭を追跡する手段を備える請求項２６７記載の装置。

【請求項２８３】　更に、非テキスト要素に識別子を付加する付加手段を備える請求項２８２記載の装置。

【請求項２８４】　非テキスト要素内に含まれる白輪郭の数に基づいて非テキスト要素を表と見なす請求項２８３記載の装置。

【請求項２８５】　更に、非格子状配置の白輪郭を再接続する手段を備え、非テキスト要素には前記再接続手段での再接続率に基づいて表の識別子が付加される請求項２８３記載の装置。

【請求項２８６】　更に、非テキスト要素内の白輪郭の充填率を算出する算出手段を備え、充填率が高くない場合に非テキスト要素に表の識別子を付加する請求項２８３記載の装置。

【発明の詳細な説明】

【０００１】

【産業上の利用分野】本発明は、文字認識方法及び装置に関わるものであり、特に認識処理に先立って、画像データに基づいて画像データブロックを分類・選別する方

し、前記追加の切出手段は前記認識手段で認識されない文字に対して行なわれる請求項２６４載の装置。

【請求項２６６】　更に、文字が前記認識手段で認識されない場合に、前記少なくとも１つの追加の切出手段で切り出された文字を再接続する第２の再接続手段を備える請求項２６５記載の装置。

【請求項２６７】　画素画像データから文字のテキストファイルを形成する装置であって、

画素画像データが２値の画素画像データでない場合に、画素画像データを２値の画素画像データに変換する変換手段を含み、画素画像データを入力する入力手段と、

画素画像データ内で連結要素の輪郭を追跡する輪郭追跡手段と、追跡された連結要素のサイズに基づいて追跡された連結要素がテキスト要素を含むか非テキスト要素を含むから決める決定手段と、隣接するテキスト要素の近接関係に基づいてテキスト要素を選択的に横方向に連結してテキストラインを形成する第１の連結手段と、隣接するテキストラインの近接関係とテキストライン間の非テキスト要素の位置とに基づいてテキストラインを選択的に縦方向に連結する第２の連結手段とを含み、画素画像データのブロックを選別する選別手段と、

前記輪郭追跡手段で追跡された連結要素に基づいて、階層的木構造を形成する形成手段と、

少なくとも１つの段の全域での画素密度の水平投影に基づいて、テキストブロックを前記少なくとも１つの段に適応的に分配することによって、テキストブロックを画素データのラインに分割する分割手段と、

非接触で重ならない文字をラインセグメントから切り出す第１の切出手段と、接触文字を切り出す少なくとも１つの追加の切出手段とを含み、前記分割手段で分割されたラインから文字を切り出す切出手段と、

前記文字切出手段で切り出された文字を認識し、前記認識に基づいて文字コードを割り付ける認識手段と、

階層的木構造により確定された順に対応して文字コードをコンピュータのテキストファイルに格納する格納手段とを備える装置。

【請求項２６８】　更に、画素画像データが前処理される前処理手段を備える請求項２６７記載の装置。

【請求項２６９】　前記前処理手段は画像圧縮手段を含む請求項２６８記載の装置。

【請求項２７０】　前記前処理手段は画素画像データの画質を向上する画質向上手段を含む請求項２６９記載の装置。

【請求項２７１】　更に、文脈チェック手段を含み、前記認識手段で認識された文字を後処理する後処理手段を備える請求項２６７記載の装置。

【請求項２７２】　前記第１の切出手段はラインセグメントをとびとびに処理する手段を含む請求項２６７記載の装置。

【請求項２７３】　前記追加の切出手段は画素画像データの文字について特徴が既知か否かに対応して選択可能である請求項２６７記載の装置。

【請求項２７４】　前記追加の切出手段は既知の文字セットの統計に基づいて切り出しを行う請求項２７３記載の装置。

【請求項２７５】　前記追加の切出手段は画素密度の回転投影に対応して切り出しを行う請求項２７３記載の装置。

【請求項２７６】　画素密度の回転投影は画素密度の垂直投影に基づいて決められた角度で得られる請求項２７５記載の装置。

【請求項２７７】　更に、前記第１の切出手段の後で前記少なくとも１つの追加の切出手段の前に中間の階層を切り出す中間の切出手段を備える請求項２６７記載の装置。

【請求項２７８】　前記中間の切出手段はラインセグメント内の重なった画素画像の輪郭を追跡する輪郭追跡手段を含む請求項２７７記載の装置。

【請求項２７９】　更に、文字が間違って切り出された場合に、前記中間の切出手段で切り出された文字を再接続する第１の再接続手段を備える請求項２７８記載の装置。

【請求項２８０】　前記認識手段は、前記追加の切出手段の前に前記第１の切出手段で切り出された文字を認識し、前記追加の切出手段は前記認識手段で認識されない文字に対して行なわれる請求項２７９載の装置。

【請求項２８１】　更に、文字が前記認識手段で認識されない場合に、前記少なくとも１つの追加の切出手段で切り出された文字を再接続する第２の再接続手段を備える請求項２８０記載の装置。

【請求項２８２】　更に、非テキスト要素の内部で白輪郭を追跡する手段を備える請求項２６７記載の装置。

【請求項２８３】　更に、非テキスト要素に識別子を付加する付加手段を備える請求項２８２記載の装置。

【請求項２８４】　非テキスト要素内に含まれる白輪郭の数に基づいて非テキスト要素を表と見なす請求項２８３記載の装置。

【請求項２８５】　更に、非格子状配置の白輪郭を再接続する手段を備え、非テキスト要素には前記再接続手段での再接続率に基づいて表の識別子が付加される請求項２８３記載の装置。

【請求項２８６】　更に、非テキスト要素内の白輪郭の充填率を算出する算出手段を備え、充填率が高くない場合に非テキスト要素に表の識別子を付加する請求項２８３記載の装置。

【発明の詳細な説明】

【０００１】

【産業上の利用分野】本発明は、文字認識方法及び装置に関わるものであり、特に認識処理に先立って、画像データに基づいて画像データブロックを分類・選別する方

法や装置に関するものである。例えば、画像データがテキスト画像データであるか、中間調（あるいはグレースケール）画像，線画及びフレームなどのような非テキスト画像であるかに応じて、画像データブロックを選別して分類するものである。また、本発明は、認識処理への準備として、テキストブロックからテキストラインを識別して分割し、次にテキストライン中の個々の文字を識別してライン中の他の文字から切り出す方法や装置にも関するものである。

【０００２】尚、本出願は付録のコンピュータプログラム（図３５～図２４７参照）とともに提出される。本明細書の開示の一部には、著作権保護に関係するものが含まれている。著作権者は、特許・商標庁の特許ファイルあるいは記録なので本出願の開示文書がファクシミリ再生されることに対しては異議を唱えないが、そうでない場合にはすべての著作権を留保するものとする。

【０００３】

【従来の技術】近年、テキストデータ画像を分析してテキストデータ中の個々の文字を認識し、認識された文字に対応してコンピュータが理解可能な文字コードファイルを生成することが可能になってきた。ここで得られたファイルは、ワードプロセッシングやデータプロセッシングのプログラムで操作を加えることができる。以下「文字認識システム」と呼ぶことにするこのようなシステムの利点は、テキストデータを再タイプしたり、あるいは再入力する必要がないことである。例えば、ファクシミリで送られてきた文書や、マイクロフィルムや複写機で再生された文書を文字認識して、文書中の文字や数字を文字コード（例えばアスキーコードコード）からなるコンピュータテキストファイルとして生成することにより、文書の再タイプや再入力をせずに、文書をワードプロセッシングやデータプロセッシングすることができる。

【０００４】文字認識すべき文書にはさまざまな種類の画像が含まれることが多く、すべてが認識可能というわけではない。例えば、テキスト画像データの認識は現在でも可能であるが、非テキスト画像データの認識は不可能である。一般に、文字認識すべき文書には、テキスト画像データブロックと中間調画像，線画及び線などの非テキスト画像データブロックとが含まれる。また、文書には、フレームで囲まれていたり囲まれていなかったりする表や表形式のデータなども含まれる。したがって、文字認識処理に先立って、ブロック中の画像データの種類に応じて文書中の個々のブロックを分類し、画像データからテキストタイプのブロックを選別することが必要となる。

【０００５】図３２は典型的な文書の１ページの例を示す図である。図３２において、文書ページ４０１は２段組のフォーマットで構成されている。ページには、タイトルに適した大きなフォントサイズのテキスト情報が含まれるタイトルブロック４０２、テキストデータラインを含むテキストブロック４０４、テキストではないグラフィックス画像を含むグラフィックスブロック４０５、テキストや数字情報の表を含む表ブロック４０６、小さなサイズのテキストデータでグラフィックスブロックあるいは表ブロックに対する見だしが含まれる見だしブロック４０７が存在する。各情報ブロックはブロック内の情報の種別に応じて分類され、この分類結果に基づいてブロックが分割される。

【０００６】画像データからテキスト型のブロックを検出するため、従来は、画像データ中の黒画素を隣接する１つあるいは複数の白画素にまで水平方向・垂直方向に膨張させて、水平方向・垂直方向に画素画像データを塗りつぶす手法を用いていた。このような塗りつぶし法の欠点としては、適切な塗りつぶしパラメータを選択するために、前もってテキスト型の画像データ中の特徴（例えばフォントサイズなど）を知らなければならないことが挙げられる。また、塗りつぶしパラメータの僅かな変更が、選別結果の大幅な変化につながるという問題もある。さらに、塗りつぶし法では、必ずしも原文書の内部構造が保存されるとは限らない。例えば、塗りつぶし法では、２段組の原文書が１段に塗りつぶされてしまうことがある。このような場合には、テキストデータの格納順序がばらばらになってしまい、原テキストの正確な再生が不可能という問題が生じる。この他、テキスト型のデータが非テキスト型のデータまで塗りつぶして、全領域をテキスト型のデータと誤って解釈してしまう場合も、塗りつぶし法ではしばしば生じる。

【０００７】ブロック選別に続いて、文書中の文字ごとに文字認識処理を行い、文字に対応するコンピュータコードを得る。ここで、文字ブロックから個々の文字を得るには、２つの処理を行う。第１段の処理では、タイトルブロック２０２，テキストブロック２０４，見だしブロック２０７などの各テキストブロック中の個々のラインが、テキストブロック中の他のラインから分割される。一般に、ライン分割は、各ブロック中の画素密度を水平投影してライン間のギャップを識別することで行われる。すなわち、図３３Ａに示すように、テキストブロック４０４は間にギャップ４１２を有するテキストラインからなる。画素密度の水平投影４１４は、ブロック４０４の各ライン上の黒画素数の総和として求められる。テキストラインは投影４１４中で密度が非ゼロの領域に対応し、テキストライン間のギャップは投影４１４中の密度がゼロの領域に対応する。したがって、投影密度に基づいて、テキストラインを各ラインごとに分割することができる。

【０００８】第２段の処理では、分割されたテキストライン中の個々の文字がテキストライン中の他の文字から切り出される。すなわち、図３４Ａに示すように、テキストライン４１１には個々の文字が含まれる。テキスト

法や装置に関するものである。例えば、画像データがテキスト画像データであるか、中間調（あるいはグレースケール）画像，線画及びフレームなどのような非テキスト画像であるかに応じて、画像データブロックを選別して分類するものである。また、本発明は、認識処理への準備として、テキストブロックからテキストラインを識別して分割し、次にテキストライン中の個々の文字を識別してライン中の他の文字から切り出す方法や装置にも関するものである。

【０００２】尚、本出願は付録のコンピュータプログラム（図３５～図２４７参照）とともに提出される。本明細書の開示の一部には、著作権保護に関係するものが含まれている。著作権者は、特許・商標庁の特許ファイルあるいは記録なので本出願の開示文書がファクシミリ再生されることに対しては異議を唱えないが、そうでない場合にはすべての著作権を留保するものとする。

【０００３】

【従来の技術】近年、テキストデータ画像を分析してテキストデータ中の個々の文字を認識し、認識された文字に対応してコンピュータが理解可能な文字コードファイルを生成することが可能になってきた。ここで得られたファイルは、ワードプロセッシングやデータプロセッシングのプログラムで操作を加えることができる。以下「文字認識システム」と呼ぶことにするこのようなシステムの利点は、テキストデータを再タイプしたり、あるいは再入力する必要がないことである。例えば、ファクシミリで送られてきた文書や、マイクロフィルムや複写機で再生された文書を文字認識して、文書中の文字や数字を文字コード（例えばアスキーコードコード）からなるコンピュータテキストファイルとして生成することにより、文書の再タイプや再入力をせずに、文書をワードプロセッシングやデータプロセッシングすることができる。

【０００４】文字認識すべき文書にはさまざまな種類の画像が含まれることが多く、すべてが認識可能というわけではない。例えば、テキスト画像データの認識は現在でも可能であるが、非テキスト画像データの認識は不可能である。一般に、文字認識すべき文書には、テキスト画像データブロックと中間調画像，線画及び線などの非テキスト画像データブロックとが含まれる。また、文書には、フレームで囲まれていたり囲まれていなかったりする表や表形式のデータなども含まれる。したがって、文字認識処理に先立って、ブロック中の画像データの種類に応じて文書中の個々のブロックを分類し、画像データからテキストタイプのブロックを選別することが必要となる。

【０００５】図３２は典型的な文書の１ページの例を示す図である。図３２において、文書ページ４０１は２段組のフォーマットで構成されている。ページには、タイトルに適した大きなフォントサイズのテキスト情報が含まれるタイトルブロック４０２、テキストデータラインを含むテキストブロック４０４、テキストではないグラフィックス画像を含むグラフィックスブロック４０５、テキストや数字情報の表を含む表ブロック４０６、小さなサイズのテキストデータでグラフィックスブロックあるいは表ブロックに対する見だしが含まれる見だしブロック４０７が存在する。各情報ブロックはブロック内の情報の種別に応じて分類され、この分類結果に基づいてブロックが分割される。

【０００６】画像データからテキスト型のブロックを検出するため、従来は、画像データ中の黒画素を隣接する１つあるいは複数の白画素にまで水平方向・垂直方向に膨張させて、水平方向・垂直方向に画素画像データを塗りつぶす手法を用いていた。このような塗りつぶし法の欠点としては、適切な塗りつぶしパラメータを選択するために、前もってテキスト型の画像データ中の特徴（例えばフォントサイズなど）を知らなければならないことが挙げられる。また、塗りつぶしパラメータの僅かな変更が、選別結果の大幅な変化につながるという問題もある。さらに、塗りつぶし法では、必ずしも原文書の内部構造が保存されるとは限らない。例えば、塗りつぶし法では、２段組の原文書が１段に塗りつぶされてしまうことがある。このような場合には、テキストデータの格納順序がばらばらになってしまい、原テキストの正確な再生が不可能という問題が生じる。この他、テキスト型のデータが非テキスト型のデータまで塗りつぶして、全領域をテキスト型のデータと誤って解釈してしまう場合も、塗りつぶし法ではしばしば生じる。

【０００７】ブロック選別に続いて、文書中の文字ごとに文字認識処理を行い、文字に対応するコンピュータコードを得る。ここで、文字ブロックから個々の文字を得るには、２つの処理を行う。第１段の処理では、タイトルブロック２０２，テキストブロック２０４，見だしブロック２０７などの各テキストブロック中の個々のラインが、テキストブロック中の他のラインから分割される。一般に、ライン分割は、各ブロック中の画素密度を水平投影してライン間のギャップを識別することで行われる。すなわち、図３３Ａに示すように、テキストブロック４０４は間にギャップ４１２を有するテキストラインからなる。画素密度の水平投影４１４は、ブロック４０４の各ライン上の黒画素数の総和として求められる。テキストラインは投影４１４中で密度が非ゼロの領域に対応し、テキストライン間のギャップは投影４１４中の密度がゼロの領域に対応する。したがって、投影密度に基づいて、テキストラインを各ラインごとに分割することができる。

【０００８】第２段の処理では、分割されたテキストライン中の個々の文字がテキストライン中の他の文字から切り出される。すなわち、図３４Ａに示すように、テキストライン４１１には個々の文字が含まれる。テキスト

25

ラインにおいて各文字を他の文字から切り出すために、ラインセグメント４１１の各列ごとに黒画素数の垂直方向の総和を求め、画素密度の垂直投影４１６を得る。文字４１５は投影４１６中で密度が非ゼロの領域に対応し、文字間のギャップは投影４１６中で密度がゼロの領域に対応する。こうして、個々の文字がラインセグメント中の他の文字から切り出される。

【０００９】このような処理においてはいくつかの問題点が存在する。例えば、文書が斜めに画像スキャナに入力され、図３３Ｂに示すように傾斜角θｓで画像メモリに格納される場合が多いが、このような場合には、第１ライン４１８のテキストが第２ライン４１９のテキストとライン４２０で重なってしまうため、必ずしもライン分割が可能であるとは限らない。すなわち、画素密度の水平投影４２１は非ゼロ値のみとなり、ゼロ値が存在しないためライン間のギャップの検出が不可能になる。

【００１０】このような問題点に鑑みて、テキストブロック４０４を図３３Ｃに示すような複数の列４２２，４２４に分割して、各列ごとに独立に水平投影を得る手法が用いられている。すなわち、図３３Ｃに示すように、水平投影４２２ａは列４２２に対応し、水平投影４２４ａは列４２４に対応するものである。各列でテキストラインが重なっていなければ、各列中のテキストラインの検出が可能となる。図３３Ｃでは２列に分割する例を示しているが、一般には５列から１０列に分割し、テストで最大傾斜角θｓまでの傾きを有していても、ブロック中の他のラインから個々のラインの分割が可能であるようにしている。

【００１１】しかし、各列ごとに水平画素投影を得なければならず、またこのようにして得た各水平画素投影も個々に処理しなければならないため、ライン分割処理がきわめて時間のかかる処理となる。また、最大傾斜角θｓの文書にまで対応しなければならないため、傾斜角が小さくて１列あるいは数列への分割で十分である多くの文書に対しても、すべての列を処理しなければならず、この点でも時間のかかる処理となっている。上記のような２段階処理のもう１つの問題は、個々の文字をラインセグメント中の他の文字から切り出す第２段の処理において生じる。

【００１２】すなわち、文字間に垂直な空間が存在する場合には、図３４Ａに示したような処理で良好な結果が得られるが、文字が垂直方向に重なっていた場合や２つあるいは複数の文字が接触している場合には、文字分割処理を適切に行うことができない。イタリックフォントの場合や、複数回の複写処理やファクシミリ送信によって画像品質が劣化している場合など、このような状況に陥る可能性は高い。例えば、図３４Ｂに示すように、イタリックテキストでは単語「Ｓａｔｉｓｆｙ」中の文字「ｆ」と「ｙ」とが垂直方向に重なっており、画素密度の垂直投影４２５が文字間でゼロにならないことがある。そのため、文字「ｆ」と「ｙ」とを分離することはできない。また、文字「ｔ」と「ｉ」も接触しているため、同様にこれら２つの文字の分離も不可能である。

26

【００１３】

【発明が解決しようとしている課題】本発明は、前記従来の欠点を除去し、高速かつ正確に文書上の文字を認識してテキストファイルを作成する文字認識方法及び装置を提供する。また、高速かつ正確に文書上のテキストと非テキストとを選別して、テキストブロックを割り出す方法及び装置を提供する。

【００１４】また、高速かつ正確に文書上の非テキストを分類する方法及び装置を提供する。また、高速かつ正確にテキストブロックからテキストラインを分割する方法及び装置を提供する。また、高速かつ正確に傾いた文書のテキストブロックからテキストラインを分割する方法及び装置を提供する。

【００１５】また、高速かつ正確にテキストラインから文字を切り出す方法及び装置を提供する。また、不適切に切り出された文字の再切り出しが可能な方法及び装置を提供する。

【００１６】

【課題を解決するための手段】この課題を解決するために、本発明の文字認識方法及び装置は以下の構成を含む。画素画像データが入力され、２値の画素画像データでない場合は２値の画素画像データに変換される。

【００１７】画素画像データのブロックが、画素画像データ内で連結要素の輪郭を追跡し、連結要素がテキスト要素を含むか非テキスト要素を含むかを連結要素のサイズに基づいて決定し、隣接するテキスト要素の近接状態に基づいてテキスト要素を選択的に連結してテキストラインを形成し、隣接するテキストラインの近接状態とテキストライン間の非テキスト要素の位置とに基づいてテキストラインを選択的に連結することにより、選別される。階層的木構造が連結要素に基づいて形成される。

【００１８】テキストブロックは、段の全域での画素密度の水平投影に基づいてテキストブロックを少なくとも１つのラインに分割することで、テキストラインに分割され、非接触で重なっていない文字を切り出す第１の切り出しと接触文字間を切り離す第２の切り出しとの２回の切り出しで、文字が分割されたラインから切り出される。切り出された文字は認識され、認識に基づいて文字コードが割り当てられる。文字コードは階層的木構造による確定された順でコンピュータのテキストファイル内に格納される。

【００１９】必要ならば、非テキスト要素は階層的木構造により確定された順に対応して格納された文字コードの間に散在してもよい。画素画像データは例えば画像圧縮や画質向上等の前処理をされ、認識された文字は例えば文脈チェック等の後処理をされてもよい。非テキスト要素の特徴に基づいて、非テキスト要素に識別子が付与

される。例えば、非テキスト要素の内部で白輪郭の追跡が行なわれ、非格子状配置の白輪郭が再連結されて、白輪郭の充填率が計算されて、白輪郭の数、非格子状配置の白輪郭の再連結率、あるいは白輪郭の充填率等に基づいて、非テキスト要素には表の識別子が付与される。

【００２０】非接触で重なっていない文字の切り出しは、ラインセグメントへのとびとびの処理で達成され、接触文字の切り離しは、接触文字間の空間に関する情報が既知か否かに対応して達成される。空間に関する情報が既知の場合は、空間の統計値に基づいて切り出しが達成される。空間に関する情報が無い場合、接触文字間の切り離しは、回転された投影によって決められた角度と位置で接触文字を斜めに切り離すように、画素密度の回転投影に対応して達成される。不適切に切り出された文字は再接続される。

【００２１】

【実施例】本発明は、複写機，ファクシミリ，ビデオカメラあるいはスチルビデオカメラ，レーザビームプリンタなどの画像処理装置あるいは画像再生装置などのような、文字認識処理が望まれるさまざまな装置において実施可能である。このような装置においては、文字画像を有する画像を、文字画像が文字認識されるように処理あるいは再生する。この際、認識した文字画像を標準文字セットあるいはフォントに替えて、原文字画像ではなく標準文字を再送あるいは再生することもできる。また、本発明は、汎用コンピュータや、パーソナルコンピュータ，ワードプロセッシングあるいはデータプロセッシング機器などのオフィス機器や、複数のビジネス機器を単一の統合パッケージにまとめた統合オフィス自動機器などにおいても実施可能である。

【００２２】図１は、スキャン，ファクシミリ，情報送受信，情報処理（ワードプロセッシングやデータプロセッシングなどの処理をも含む）などの機能を有する統合オフィス自動機器として、本発明の一実施例を示すブロック図である。図１に示す装置において、画像の入力は、ファクシミリ送信による入力、原文書のスキャンによる入力、モデムを介して離れた地点からの入力などで行われる。本実施例によれば、画像中の文字を認識し、認識文字のコンピュータテキストファイルを生成し、装置内のワードプロセッシング，スプレッドシート処理ならびに他の情報処理機能を利用してテキストファイルの修正を行うことができる。この処理に続いて、修正されたテキストファイル（あるいは無修正のテキストファイル）の再送や、音声合成技術を用いてテキストファイルのスピーカや通常の音声電話への「話しかけ」などによる出力が行われる。

【００２３】図１において、プログラム可能なマイクロプロセッサなどの中央処理ユニット（ＣＰＵ）１０はバス１１に接続されている。また、画像を画素ごとにスキャンして画像メモリ（例えば、以下で示すＲＡＭ２０）に蓄積するスキャナ１２、デジタルデータを電話線１５ａを通してアナログ形式で送受信するモデム１４、画像データを電話線１５ｂを通して送受信するファクシミリ１６（不図示の電話を含むこともある）なども、バス１１に接続される。なお、電話線１５ａと１５ｂとは同一の線であって、不図示のネットワーク制御ユニットで制御されるものでも良い。さらに、バス１１には、ＣＰＵ１０によって実行される１つあるいは複数のコンピュータプログラムを蓄えるリードオンリメモリ（ＲＯＭ）１７、認識処理において入力文字と比較する文字の辞書を保持している文字辞書１９、入力画像データ，処理画像データならびに画像の構造などの情報を蓄えるランダムアクセスメモリ（ＲＡＭ）２０、文字認識処理において文字の認識結果を出力する出力装置２１（ディスクメモリ，スピーカあるいは音声電話線インタフェースを有する音声合成装置など）、装置によって処理された画像を表示するプリンタ／ディスプレイ２２、オペレータが装置を操作するためのキーボード２４なども接続される。

【００２４】なお、ここでは、バス１１に接続された装置が協調して統合オフィス自動機器を構成しているが、これらの装置のうちのいくつかあるいはすべてが単独で用いられることも可能なことは明らかであろう。スキャナ１２，モデム１４及びファクシミリ１６は、画像データを装置に入力するための種々の入力手段である。スキャナ１２では、原画像はラインごと画素ごとにスキャンされ、ＣＰＵ１０の制御のもとで画像データの画素はＲＡＭ２０中の画像メモリにビットマップ形式で蓄えられる。モデム１４では、電話線１５ａを介してアナログ形式で受信した画像データがデジタル画素形式に変換され、デジタル画素データはＲＡＭ２０中の画像メモリに蓄えられる。ファクシミリ１６では、電話線１５ｂを介して修正ハフマンランレンクス符号などの符号形式で受信した圧縮画像データを、ファクシミリ１６において従来手法でデジタル画像の画素データに復号して、ＣＰＵ１０によって画像データの画素はＲＡＭ２０中の画像メモリにビットマップ形式で蓄えられる。ここで、他の画像入力手段を用いることはもちろん可能で、ディスクメモリなどの大容量の蓄積メディアから画像データを取り出したり、ビデオカメラあるいはスチルビデオカメラから入力することもできる。

【００２５】ファクシミリ１６や出力装置２１は、文字認識された画像データを装置から出力するための種々の手段である。ファクシミリ１６では、本実施例に基づき認識処理された文字画像が標準文字セットあるいはフォントに変換され、装置から送信される。これにより、例えば、文字画像を含む画像を受信して、文字画像を文字認識し、再送に先立って認識文字を標準文字フォントに変換することで、劣化画像の品質の改善を図ることができる。

【００２６】モデム１４や出力装置２１は、画像データ

29

で認識された文字をＡＳＣＩＩコードなどで出力あるいは蓄積するための種々の手段である。すなわち、文字は装置（ディスクメモリなど）に蓄えられたり、モデム１４を介して離れた地点に転送するために出力される。この際、ＡＳＣＩＩコードをファクシミリ互換のフォーマットに変換するなどの文字変換手段を設ければ、ファクシミリ１６を用いることなくモデム１４を介して離れた地点のファクシミリ機に転送を行うことができる。

【００２７】プリンタ／ディスプレイ２２は、本実施例に基づく文字認識処理の流れの監視、ならびに文字認識の各ステップの永久記録の出力および表示を行うための手段である。キーボード２４は、図１の装置の動作に対するオペレータの制御を可能にしている。図２は、本実施例の文字認識処理のフローチャートである。図２に示した処理ステップは、プログラムＲＯＭ１７に蓄えられているコンピュータプログラムに基づいてＣＰＵ１０で実行される。

【００２８】ステップＳ２０１において、画素単位の画像データ（以下画素画像データ）は装置に入力され、ＲＡＭ１７に蓄えられる。画像データは画像を画素ごとに表現したものである。この際、画素データは２値画素データ、すなわち黒ならびに白画像データであることが好ましい。なお、画像データは、複数のグレイ階調値の１つで各画素が表現される中間調画像であっても、画素色を表わす複数ビットワードで各画素が表現されるカラー画像データであっても良い。このような場合、すなわち画素データが２値画素データでない場合には、非２値画素データを２値画素データに変換するためのしきい値処理をＲＡＭ２０に蓄積する前に行う必要がある。

【００２９】ステップＳ２０１で入力された画素画像データは、左上隅から右下隅に読み進められるようなポートレイト画像であることが好ましい。画像がポートレイト画像でない場合、例えばランドスケイプ画像である場合には、画素画像データをポートレイト型の画像に変換する必要がある。この変換処理は、オペレータがキーボード２４から画像データの変換を指示することで行うことができる。

【００３０】ステップＳ２０２では、画像データの前処理が行われる。一般に、前処理フィルターを用いて画像データの画質を向上し、劣化した文字や画像を改善する。画質向上手法として適切な手法は、１９９１年１０月４日出願の米国出願番号０７／７７１，２２０に示されている。尚、ステップＳ２０２では、精度の劣化は避けられないが、画素画像データ中の複数の画素を削除したり圧縮したりすることで、認識処理の高速化を図ることも可能である。例えば、ｍ×ｎ画素ブロックの画素の平均値を求め、この平均値でｍ×ｎ画素ブロックを代表させることも可能である。

【００３１】ステップＳ２０３では、ブロック選択を行って各セグメント画像中の画像データの型を調べ、ブロック中の情報がテキスト情報か，グラフィックス情報か，線画情報か，画像情報かなどを識別する。また、ブロック選択ステップＳ２０３では、画像中の各部分を以下のステップＳ２１２で示すような適切な順序で再生することができるように、画像の階層的木構造をも生成する。この階層的木構造には、例えば、２段組の画像データにおいて第１段のテキストから第２段のテキストに飛んで読み進めることのないように、第１段のテキストの後に第２段のテキストを再生できるような情報が含まれる。なお、ステップＳ２０３におけるブロック選択処理については、以下でより詳細に説明する。

30

【００３２】ＲＡＭ２０に蓄えられた画像から第１の情報ブロックが選択され、ブロック選択ステップＳ２０３で判別されたブロック識別子に応じて、ステップＳ２０４において選択されたブロックがテキストブロックであるかどうかが調べられる。第１ブロックがテキストブロックでない場合には、処理をステップＳ２０５に進め、次のブロックを選択してステップＳ２０４に戻る。

【００３３】ステップＳ２０４においてブロックがテキストブロックである場合には、処理をステップＳ２０６に進め、テキストブロックに対してライン分割処理を行う。ライン分割処理では、テキストブロック中の個々のテキストラインをテキストブロック中の他のテキストラインから分離して、以下に詳細に説明するように分離したラインごとに順々に処理を行う。ステップＳ２０７では、ライン中の各文字をライン中の他の文字から切り出し、以下に詳細に説明するように個々の文字に対して文字認識処理を行う。

【００３４】ステップＳ２０８では、文字ごとに各文字の認識処理が行われ、既知の手法で文字辞書１９に蓄えられている標準文字と各文字との比較処理が行われる。文字辞書１９中の索引は一般には１つの文字に対応するが、分離が難しいような文字列（例えば、「ｆｌ」）や分離されやすい１文字（例えば「ｊ」）に対しては他の索引が用意されることもある。すなわち、分離が難しいような接触文字ペアを辞書の索引とすることも、分離されやすいような１文字中の部分を辞書の索引とすることもある。比較処理に基づいて文字画像に対する識別子が選択され、選択された文字識別子はＲＡＭ２０に蓄えられたり、出力装置２１に出力される。また、識別された文字をプリンタ／ディスプレイ２２に表示することもできる。

【００３５】ステップＳ２０９では、テキストブロックに対する処理が終了したかどうかが調べられる。処理が終了していなければ、処理をステップＳ２０６（あるいは適宜ステップＳ２０７）に戻し、ライン分割処理，文字切り出し処理及び文字認識処理とを繰り返し行う。ブロック処理が終了していれば、処理をステップＳ２１０に進め、１ページ分の処理を終了したかどうかを調べる。１ページ分の処理が終了していなければ、処理をス

31

テップS２０５に戻し、ページ中の次ブロックを選択して処理を続ける。

【００３６】１ページ分の処理が終了していれば、処理をステップS２１０からステップS２１１に進め、後処理を実行する。後処理では、文脈チェックやスペルチェックなどの処理を行い、文字認識ステップS２０８で認識された文字識別子を、ステップS２０８のような個々の文字ごとではなく文字の周囲の文脈に基づいて、全体的な見地から修正する処理を行う。ステップS２１２では、ブロック選択ステップS２０３で定義された階層的木構造に基づいて画像データが再生される。ページ再生では画像データが適切な順序で配置される。例えば、脚注はメインテキストから分離され、段は他の段と混ざることなく順番に配置され、グラフィックスや線画はページ中の認識文字テキストの適当な位置に挿入される。このようなグラフィックス画像や線画の見だしも図に隣接して挿入される。この際、他の規則を用いることもでき、例えばページの物理的再生ではなく、ページからテキストのみを抽出するという規則を用いることもできる。なお、このようなページ再生の規則は、装置の情報処理機能を用いてユーザが定義できる。

【００３７】ステップS２１３では、再生されたページが出力装置２１などに蓄えられる。ここでは、ROM１７に蓄えられており、CPU１０で実行される他の応用プログラムを用いて、スプレッドシートやワードプロセッシング処理などの情報処理を行うことができる。そして、処理された情報（あるいは適宜処理していない情報）は、ファクシミリ１６やモデム１４などを用いて、またはコンピュータテキストファイルを「しゃべる」音声合成装置を用いた通常の音声電話を介してなどのさまざまな手段を用いて再送される。

【００３８】図３は、図２のステップS２０３のブロック選別処理を説明するための一般化した処理流れ図である。ここで、ブロック選別処理は文字認識システムとは別に利用可能であることに注意されたい。すなわち、画像再生装置においてブロック選別処理を用いて、ある種のブロックに対しては画像再生の第１の手法を施し、別の種類のブロックに対しては画像再生の第２の手法を施すといったことも可能である。

【００３９】ブロック選別の処理速度の向上を望む場合には、ステップS３００で画像データを縮小する処理を行う。画像データの縮小処理を行った場合には、ブロック選別処理は縮小画像に対して行われる。なお、この際、図２の残りの文字認識処理（ステップS２０４からステップS２１３）に影響を及ぼさないように、ブロック選別処理終了時には、縮小前の画像データに対して選択ブロックが割り当てられる。画像データ縮小処理は、m×m画素ブロック中の黒画素の連結性を調べながら行われる。例えば、３×３画素ブロック中に連結された黒画素が２個ある場合には、３×３画素ブロックを１つの黒画素に縮小する。逆に、３×３画素ブロック中に連結された黒画素が２個ない場合には、３×３ブロックを１つの白画素に縮小する。

32

【００４０】ステップS３０１では、画素画像を解析して連結された要素を検出し、連結要素をサイズや他の連結要素との相対的な位置に応じて分類する処理を行う。ここで、連結要素は白画素に完全に囲まれた黒画素のグループである。すなわち、連結要素は、少なくとも１つの白画素で他の黒画素グループと完全に分離されている黒画素グループである。図４を参照して以下で詳細に説明するように、ステップS３０１では、連結要素を検出して、連結要素から得たサイズ情報やいくつかの統計的な値に基づいて各連結要素の分類を行う。

【００４１】まず、以下で詳細に説明するように、各連結要素をテキスト要素か非テキスト要素かに分類する。さらに、非テキスト部については、フレームデータであるか，中間調画像であるか，線画であるか，表あるいはテキストデータを表のように構成したものであるか，あるいは未知の要素で分類不可能であるかなどを解析処理して決定する。また、各連結要素ごとに階層的木構造を構成して、連結要素の構造的な記述データを求め、ステップS２１２で説明したようにデジタルデータの再生を容易にする。

【００４２】ステップS３０２では、水平方向に近い位置関係にあり、ギャップラインマーカを横切らないような連結要素をラインとしてグループ化する。この際、ステップS３０１で作成した木構造を用いて、テキスト要素と非テキスト要素とが不適切にグループ化されないようにする。また、ステップS３０２では、段間で垂直方向に伸びるギャップや非テキスト要素の垂直方向に伸びる境界を検出して、テキスト要素を段ごとにまとめる。ここで得られた段構造は階層的木構造に組み込まれ、適宜木構造の更新が行われる。

【００４３】ステップS３０３では、ステップS３０２でグループ化されたラインの内，垂直方向に近接するラインを垂直方向にグループ化して、ブロックを形成する。ここで、非テキスト要素は垂直方向にソートされ、画像ページの境界として用いられる。なお、２つの非テキスト要素の間に位置するテキストライン要素は、他のテキストライン要素とは別に処理される。また、ステップS３０３では、ステップS３０１で分類できなかった非テキスト要素が、大きなフォントサイズのタイトルであるかどうかを判別する。タイトルであると判別されると、その部分に「タイトル」属性を付与し、木構造を更新する。ここで得られたタイトルは、ステップS２１２に基づくページの再生の際に有用な情報となりうる。

【００４４】図４A～図４Cは、画素画像データの連結要素をどのように検出して、どのようにこれらの連結要素を分類するかを示す詳細なフローチャートである。図４A～図４Cに示される処理ステップは、ROM１７に

保持されているプログラムステップに基づいてＣＰＵ１０で実行される。ステップＳ４０１では、輪郭追跡を行い画素画像データの連結要素を検出する。輪郭追跡処理は図５Ａに示すように画像データをスキャンすることで行われる。スキャン処理は矢印Ａで示されるように画像の右下部から左に行われ、画像の右側の端に達したときに上に進むように行われる。なお、この際、逆方向すなわち左上から右下にスキャンすることもできる。スキャン処理中に黒画素を検出すると、３１で示すような星形パターンの順序で隣接画素を調べ、黒画素の隣接画素もまた黒画素であるかどうかを判別する。星形パターン３１には８個の番号付けされたベクトルが中心点から伸びているため、このような輪郭追跡を以下「８方向」追跡と呼ぶことにする。黒画素が隣接して存在する場合には、画像の外側輪郭を追跡し終えるまで上述の処理を繰り返す。

【００４５】すなわち、図５Ｂに示すように、矢印Ａ方向へのスキャン処理によって、まず文字「Ｑ」の端に対応する点３２が検出される。次いで、星形パターン３１に基づいて隣接画素が判別され、文字「Ｑ」の外側輪郭が追跡される。この際、閉輪郭の内部の輪郭追跡処理は行わない。連結要素を１つ検出して、８方向追跡で輪郭の追跡を終えると、次の黒画素を検出するまでスキャン処理を続ける。すなわち、完全な黒領域である物体３４の８方向追跡を行う。また、手書き単語「ｎｏｎ－ｔｅｘｔ」の非テキスト物体３５を、単語「ｗｏｒｄ」を構成する個々の文字のテキスト物体３６中の個々の文字と同様に追跡する。図５Ａに示すスキャン処理は、画素データ中のすべての連結要素を検出し、８方向追跡して輪郭を検出するまで行われる。

【００４６】次いで処理をステップＳ４０２に進め、各連結要素を矩形で囲む処理を行う。具体的には、各連結要素の周りを包囲する最も小さな矩形を描く。すなわち、図５Ｂに示すように、物体３２の周りには矩形３７が、物体３４の周りには矩形３９が、物体３５の周りには矩形４０が、テキスト物体３６ａ，３６ｂ，３６ｃ，３６ｄの周りには矩形４１ａ，４１ｂ，４１ｃ，４１ｄが描かれる。

【００４７】ステップＳ４０３では、各矩形要素に対して木構造中の位置が割り当てられる。ほとんどの場合、ステップＳ４０３で得られる木構造では、画素画像中の各物体は木の根に直接つながる。これは、連結要素の外側輪郭のみを追跡しており、閉輪郭の内部は追跡していないためである。すなわち、図５Ｃに示すように、連結要素３２に対応する矩形３７はページの根に直接つながる。これに対し、非テキスト物体３５に対応する矩形４０や、テキスト物体３６ａ，３６ｂに対応する矩形４１ａ，４１ｂなどのように、矩形が他の矩形の内部に完全に位置するような連結要素は、囲っている連結要素（この場合には要素３４）の子ノードとして位置付けられる。また、要素３４などのように少なくとも１つの子を有する連結要素は、それ自身を「主な子ノード」として位置付ける。すなわち、図５Ｃに示すように、要素３９が「主な子ノード」として、要素３９の他の子ノード４０，４１ａ，４１ｂとともに位置付けられる。

【００４８】ステップＳ４０４では、木構造の第１レベルの各連結要素をテキスト要素あるいは非テキスト要素に分類する。この分類処理は２つのステップで行われる。第１ステップでは、連結要素の矩形と所定のしきい値サイズとの比較を行う。連結要素を囲む矩形の高さが、想定される最大フォントサイズに対応する所定の第１しきい値以上であれば、あるいは連結要素を囲む矩形の幅が、ページ幅を経験的に決められた一定値で割った値以上であれば（「５」で良好な結果が得られることがわかっている）、連結要素を非テキスト要素と分類し、要素に「非テキスト」の属性を付与する。

【００４９】第２ステップでは、残りすべての要素すなわち非テキストと分類されなかった要素と、残りすべての連結要素のサイズの集合に基づいて適当に求められるしきい値との比較を行う。具体的には、非テキスト要素として分類されなかったすべての矩形の平均の高さを求め、この平均高にスカラー値を掛け合わせたものを（「２」を用いると好都合である）適当に求められるしきい値とする。そして、適当に求めたしきい値以上に大きな要素はすべて非テキストであると考え、非テキスト要素として分類する。また、適当に求めたしきい値以下の小さな要素はすべてテキスト要素であると考える。このようにして要素の分類を行い、適切な属性が付与される。

【００５０】なお、図４Ａ～図４Ｃの残りを参照して以下で詳細に説明するように、ここで求めた分類結果は以下で改善される。木構造の第１レベルの各要素をテキスト要素あるいは非テキスト要素として分類すると、テキスト要素の主な子ノードを含むすべての子ノードをテキスト要素として分類する。これに対して、非テキスト要素の主な子ノードは非テキスト要素として分類するが、非テキスト要素の他のすべての子ノードはテキスト要素と分類する。

【００５１】ステップＳ４０５において、最初の要素を選択する。要素がテキスト要素であれば（ステップＳ４０６）、ステップＳ４０７に処理を進め次の要素を選択する。非テキスト要素を選択して処理をステップＳ４０８に進めるまで、ステップＳ４０６とＳ４０７の処理を繰り返す。ステップＳ４０８では、非テキスト要素に子ノードが存在するかどうかが調べられる。例えば、図５Ｃに示されるように、非テキスト要素３９には、非テキストの主な子ノード３９とテキストの子ノード４０，４１ａ，４１ｂとが存在する。

【００５２】ステップＳ４０８において子ノードが存在する場合には、処理をステップＳ４０９に進め、要素に

35

フィルタ処理が施され、要素が中間調（あるいはグレイ階調）要素であるかどうかが判別される。中間調フィルタ処理では要素の子ノードを調べ、要素の子ノードのうち「雑音サイズ」以下であるようなサイズのものを求める。ここで「雑音サイズ」の要素とは、高さが画像データにおいて想定される最小のフォントサイズ以下であるような要素のことである。雑音サイズ以下のサイズの子ノード数が全体の子ノード数の半分以上であれば、要素は中間調画像として判断される。そこで、ステップＳ４１０で処理をステップＳ４１１に進め、「中間調」属性を要素に付与する。

【００５３】次いでステップＳ４１２で、中間調画像中のテキスト要素をチェックする。具体的には、木構造中の中間調画像の子ノードのうちテキストサイズの子ノードを修正して、テキストサイズの要素を中間調画像の子ノードとしてではなく、中間調画像と同一レベルのノードに配置する。これにより、中間調画像中のテキストサイズの要素に対しても、適切だとすれば文字認識処理を行うことができる。この処理を終えると処理をステップＳ４０７に戻し、次の要素を選択して処理を行う。

【００５４】ステップＳ４０９の中間調フィルタ処理において要素が中間調画像でないと判別された場合には、処理をステップＳ４１０からステップ４１３に進め、さらなる処理を施すために要素の主な子ノードを選別する。そして、処理をステップＳ４１４に進める。ステップＳ４０８において非テキスト要素が子ノードをもたないと判別される場合、あるいはステップＳ４１３においてさらなる処理を施すために主な子ノードが選択されると、ステップＳ４１４で当該要素に対してフレームフィルタ処理が施される。フレームフィルタ処理は、当該要素がフレームであるかどうかを調べ、当該要素を囲むような矩形とほぼ同一の幅／高さを有する平行水平線と平行垂直線とを検出するための処理である。具体的には、画素中の行ごとに当該要素中の連結要素内部の最大長を求めて、連結要素が調べられる。

【００５５】すなわち、図６Ａに示されるように、非テキスト要素４２には連結要素４３が含まれるが、その輪郭は８方向追跡によって４４で示すように追跡処理される。行「ｉ」における連結要素内部の最大長は、輪郭の左端４５ａから右端４５ｂまでの距離x_iとなる。また、行「ｊ」では、連結要素内部の長さとして、連結要素の境界上の点４６ａと４６ｂの距離、ならびに点４７ａと４７ｂの距離との２つが存在する。ここで、点４６ａと４６ｂの距離の方が点４７ａと４７ｂの距離よりも長いため、距離x_jが行ｊにおける連結要素内部の最大長となる。

【００５６】非テキスト要素４２中のｎ個の行ごとに距離「ｘ」を求め、以下の不等式を満たすかどうかを調べ、非テキスト要素がフレームであるかどうかを判別する。

36

【００５７】

【数１】

$$\sum_{k=1}^{N}\frac{(X_k-W)^2}{N} < threshold \quad (しきい値)$$

ここで、x_kは（上述のように）連結要素内部のｋ行目の最大長，Ｗは矩形要素４２の幅，Ｎは行数，しきい値はフレームが画像データ中で傾いていてもフレーム検出が可能となるように前もって計算した値である。ここで、１度の傾き角を許容するには、しきい値としてｓｉｎ（１°）をＬ倍したものに、ステップＳ４０４で計算したテキストの平均の高さをオフセットとして加えたものを用いると良い。

【００５８】上記の不等式が満たされれば、要素はフレーム要素であると判断され、ステップＳ４１５からステップＳ４１６に処理を進め、「フレーム」属性を要素に付与する。ここで、複数の属性が各要素に付与されうることに注意されたい。すなわち、「フレーム－表」、「フレーム－中間調」などのようにフレームの属性が付与されることもある。

【００５９】ステップＳ４１６に続いて、フレーム要素中に表データあるいは表のように組織されたデータが存在するかどうかを調べる処理を行う。そこで、ステップＳ４１７において、連結要素の内部を調べて白輪郭を求める。白輪郭は、黒画素ではなく白画素に着目するという点を除けば、上のステップＳ４０１で検出した輪郭と同一のものである。すなわち、図７Ａに示されるように、非テキスト要素の内部を非テキスト要素の右下から左上へ矢印Ｂの方向に沿ってスキャンする。このスキャン処理によって第１の白画素が検出されると、星形パターン５１の順序で白画素の隣接画素がチェックされる。星形パターン５１には１から４までの数が付与されたベクトルがある。このため、このような処理に基づく白輪郭追跡処理のことを以下「４方向」白画素追跡と称する。

【００６０】黒画素に囲まれた白輪郭すべてを追跡し終えるまで、４方向の白画素追跡処理を繰り返す。例えば、白輪郭追跡処理によって、黒画素セグメント５２，５３，５４，５５ならびに内部に存在する５６のような他の黒画素で構成される内部輪郭の画素が追跡される。このようにして白輪郭を位置付けると、矢印Ｂ方向へのスキャン処理を、非テキスト物体に囲まれたすべての白輪郭を追跡するまで繰り返す。

【００６１】ステップＳ４１８では、非テキスト要素の密度を計算する。密度は連結要素中の黒画素数を計数して、黒画素数を矩形中の全画素数で割ることで計算される。ステップＳ４１９では、非テキスト要素中で検出された白輪郭数の数をチェックする。白輪郭の数が４以上であれば、非テキスト画像が表またはテキストブロックを表のように並べたものである可能性が高い。

【００６２】そこで、ステップＳ４２０において、白輪郭の充填率を求める。白輪郭の充填率は、白輪郭が非テキスト画像中の領域を占める割合のことである。例えば、図７Ａに示されるように、白輪郭充填率には、５７や５９のような斜線領域で完全に空の白領域と、６０や６１のような領域で中に黒画素を含むような白領域とが含まれる。この充填率が高いと、非テキスト画像が表またはテキストデータを表のように並べたものである可能性が高い。そこで、ステップＳ４２１において充填率をチェックする。この際、高い充填率であれば、非テキスト画像は表またはテキストデータを表のように並べたものであると考えられる。

【００６３】この処理の信頼性を向上させるために、白輪郭が水平ならびに垂直方向に格子状の構造をなしているかどうかを調べる。すなわち、ステップＳ４２２において、少なくとも２つの白輪郭の境界線が水平方向ならびに垂直方向に一致しないような非格子状の白輪郭を再接続する。例えば、図７Ａに示されるように、白輪郭５９の左境界線６２及び右境界線６３は、白輪郭６０の左境界線６４及び右境界線６５と垂直方向に一致する。そのため、これらの白輪郭は格子状に配置されており、白輪郭の再接続は行わない。同様に、白輪郭５９の上部境界線６６及び下部境界線６７は、白輪郭７０の上部境界線６８及び下部境界線６９と水平方向に一致する。そのため、これらの白輪郭は格子状に配置されており、これらの白輪郭の再接続は行わない。

【００６４】図７Ａから図７Ｄまでは、白輪郭を再接続する場合を説明するための図である。図７Ｂは非テキスト要素７１を示しており、これはステップＳ２０１で示した中間調画像の２値画像へのしきい値処理などによって得られるものである。非テキスト画像７１には、黒領域７２と白領域７４，７５，７６，７７，７８，７９が存在する。ここで、これらの白領域の充填率は十分高く、ステップＳ４２１において処理は再接続ステップＳ４２２に進められたものとする。

【００６５】まず、図７Ｃに示されるように、白輪郭７５の左境界線及び右境界線と、白輪郭７６の左境界線及び右境界線とを比較する。すると、これらの境界線は一致しないため、白輪郭７５と白輪郭７６とは再接続されて図７Ｃの接続白輪郭７６'が生成される。次いで、図７Ｄに示されるように、白輪郭７７の上部境界線及び下部境界線と、白輪郭７９の上部境界線及び下部境界線とを比較する。すると、これらの境界線は一致しないため、白輪郭７７と白輪郭７９とは１つの接続白輪郭７７'として結合される。再接続される輪郭がなくなるまで、これらの処理を水平方向に垂直方向に繰り返す。

【００６６】すなわち、上述のように、表をなす白輪郭は再接続されにくいのに対し、中間調画像や線画などの表でないような白輪郭は再接続されやすい。そこで、ステップＳ４２３において接続率を調べる。ここで、再接

続率が高ければ、あるいは再接続処理後に残った白輪郭の数が４より少なければ、処理をステップＳ４２８に進め、以下で詳述するように、非テキスト要素に「中間調画像」もしくは「線画」の属性を付与する。ステップＳ４２３において再接続率が高くなく、且つ少なくとも４つの白輪郭が残っていれば、処理をステップＳ４２４に進め、非テキスト画像に「表」の属性を付与する。

【００６７】ステップＳ４２５では、新たに属性が付与された表の内部を調べ、８方向追跡処理で連結要素を検出し分類する。ステップＳ４２６では、この新たな内部の連結要素に基づいて階層的木構造を更新する処理を行う。続くステップＳ４２７では、ステップＳ４０２からステップＳ４０４で示したような手法でもって、内部連結要素をテキスト要素もしくは非テキスト要素に再分類し、適当な属性を付与する。これらの処理を終えると、処理をステップＳ４０７に戻し次の要素が選択される。

【００６８】ステップＳ４２１とＳ４２３に戻って、ステップＳ４２１で充填率が高くない場合、あるいはステップＳ４２３で再接続率が高い場合には、非テキストフレーム要素は中間調画像もしくは線画である可能性が高いと考えられる。ここで、要素を中間調画像として分類するか、あるいは線画として分類するかの決定は、要素中の黒画素の平均水平ラン長、要素中の白画素の平均水平ラン長，白画素と黒画素との比率，及び密度に基づいて行われる。一般に、暗めの画像は中間調画像であると考えられ、明るめの画像は線画であると考えられる。

【００６９】具体的には、白画素の平均水平ラン長がほぼゼロ（すなわち、全体が暗めあるいはまだらな画像）であって、ステップＳ４１８で求めた密度によって要素が白っぽいというよりもむしろ黒っぽい場合（すなわち、密度が１／２程度の第１しきい値より高い）には、フレーム要素は中間調画像であると判別される。密度が第１しきい値以下である場合には、要素は線画として判別される。

【００７０】白画素の平均水平ラン長がほぼゼロではなく、黒画素の平均水平ラン長より長い場合には、フレーム要素は線画であると判別される。しかし、白画素の平均水平ラン長が黒画素の平均ラン長以下（すなわち、暗めの画像）である場合には、さらなる判別処理が必要となる。具体的には、黒画素数が白画素数よりもかなり少ない（すなわち、黒画素数を白画素数で割った値が約２の第２しきい値より大である）場合には、フレーム要素を中間調要素として判別する。これに対し、黒画素数を白画素数で割った値が第２しきい値以下であっても、ステップＳ４１８で求めた密度が第１しきい値より高ければ、フレーム要素を中間調画像として判別する。これ以外であれば、フレーム要素は線画として判別される。

【００７１】ステップＳ４２８においてフレーム要素が線画として判別されれば、処理をステップＳ４２９に進めて「線画」属性を付与し、ステップＳ４３０ですべて

39

の子ノードを削除する。なお、要素が線画として判別されると、線画要素のいかなるブロックも文字認識のために選択されることはない。そして、処理をステップＳ４０７に戻し次の要素を選択する。

【００７２】一方、ステップＳ４２８においてフレーム要素が線画として判別されなかった場合には、処理をステップＳ４３１に進めて「中間調」属性を付与し、ステップＳ４３２でフレーム中間調要素のテキストサイズの子ノードを削除する。ここで、テキストサイズは、ステップ４０４において述べたように平均の要素の高さとして求められる。また、テキストサイズより大きなすべての子ノードは、フレーム中間調要素の子ノードのままとする。これらの処理を終えると、処理をステップＳ４０７に戻し次の要素を選択する。

【００７３】ステップＳ４１９に戻り、白輪郭の数が４より小さい場合には、フレーム要素は表であるとは判別されない。そこで、処理をステップＳ４３３に進め、ステップＳ４１８で求めた密度と約０．５のしきい値とを比較する。ここで、しきい値は、フレーム内のテキスト要素や線画が占める面積は画素の半分以下であることを鑑みて選択されている。

【００７４】密度がしきい値より小さければ、処理をステップＳ４３４に進め、上述のようにフレーム要素の内部構造を作成する。すなわち、処理をステップ４０１に戻し、フレーム要素の内部構造に対する処理を行う。ステップＳ４３３において、密度が所定のしきい値以上であった場合には、処理をステップＳ４４２に進め、フレーム要素を線画，中間調画像あるいは分類不可能（すなわち、フレームが「未知」）のどれかに分類する。

【００７５】ステップＳ４１５に戻り、ステップＳ４１４におけるフレームフィルタ処理で非テキスト要素中にフレームが検出されなかった場合には、処理をステップＳ４３５に進め、非テキスト要素中に線が含まれるかどうかを調べる。線は、テキスト境界を明確にするのに適した非テキスト要素である。この際、このような線に囲まれたテキストは線のそばに位置することが多いため、テキストが線に接するということもありうる。そのため、テキストが接していても接していなくても線を検出できるような線の検出処理を行う。

【００７６】テキストが接していない線を検出するには、非テキスト要素の長さ方向のヒストグラムを求める。すると、図６Ｂに示されるように、線のヒストグラム４８はほぼ均一な分布となり、ヒストグラムの高さは線幅にほぼ等しくなる。また、線幅は、非テキスト要素の幅「W」にほぼ等しい。ここで、線幅と非テキスト要素の幅との差は、画素画像を生成する際の原文書の傾き角θｓによるものである。そこで、非テキスト要素が線を含むかどうかを調べるために、ヒストグラム中の各セルｋの高さ４９を非テキスト要素の幅Wと比較する。すなわち、これらの値間の実効値誤差としきい値とを次式

40

のように比較する。

【００７７】

【数２】

$$\sum_{k=1}^{N}\frac{(cell_k-W)^2}{N} < threshold \quad (しきい値)$$

ここで、しきい値は、非テキスト要素中の線の傾き角θｓを許容できるように設定される。例えば１°の傾き角であれば、しきい値として

【００７８】

【数３】

$$\sum_{k=1}^{N}\left[\frac{k\sin(1°)}{N}\right]^2$$

を用いると良好な結果が得られる。上の不等式によってテキスト要素が接していない線が検出されなかった場合には、テキスト要素が接している線を含む要素であるかどうかを調べる処理を行う。テキスト要素に接している線が非テキスト要素中に含まれるかどうかを判別するために、要素の境界に沿って線が長く伸びているかどうかを調べる。すなわち、要素を通して線が長く伸びていれば、図６Ｃに示すように要素を囲む矩形の境界は線にきわめて近いところに位置する。そこで、矩形の境界近くの第１番目の黒画素位置の均一性を、境界からの距離の２乗の和を求めることで判断する。すなわち、次式の不等式を満たすかどうかを調べる（図６Ｃ参照）。

【００７９】

【数４】

$$\sum_{k=1}^{N}\frac{X_k^2}{N} < threshold \quad (しきい値)$$

ここで、２乗の和が所定のしきい値より小である場合には、テキスト要素が接している線要素と判別する。この際、しきい値として、テキスト要素が接していない線の検出におけるいき値を用いると、良好な結果が得られる。ステップＳ４３５において線を検出すると、処理をステップＳ４３６からステップＳ４４９に進め、「線」属性を非テキスト要素に付与する。そして、処理をステップＳ４０７に戻し、次の要素を選択する。

【００８０】一方、ステップＳ４３５で線が検出されなかった場合には、処理をステップＳ４３６からステップＳ４３７に進め、非テキスト要素のサイズを調べる。ここで、サイズが所定のしきい値以下であると、非テキスト要素の分類が不可能となる。なお、このしきい値は最大のフォントサイズに基づいて決められるものであり、最大フォントサイズの半分の値で良好な結果が得られる。そして、処理をステップＳ４３８に進めて「未知」属性を非テキスト要素に付与し、処理をステップＳ４０７に戻して次の要素を選択する。

【００８１】ステップＳ４３７においてサイズが所定の

しきい値よりも大であった場合には、処理をステップＳ４３９，Ｓ４４０，Ｓ４４１に進め、ステップＳ４１７，Ｓ４１８，Ｓ４１９で述べたように、非テキスト要素内部の白輪郭を追跡し、非テキスト要素の密度を求め、白輪郭の数を調べる。ステップＳ４４１において白輪郭の数が４以下であれば処理をステップＳ４４２に進め、要素サイズを計算し、要素が線画や中間調画像であるのに十分な大きさであるかを調べる。このサイズ計算は、非テキスト要素の高さと幅ならびに黒画素の最大ラン長に基づいて行われる。具体的には、非テキスト要素の高さや幅が最大フォントサイズ以下であれば、非テキスト要素は中間調画像や線画であるほど大きくはないと判断し、処理をステップＳ４４３に進め「未知」属性を付与する。また、非テキスト要素の幅は最大フォントサイズより大であるが、黒画素の最大ラン長は最大フォントサイズ以下である場合にも、処理をステップＳ４４３に進め、「未知」属性を付与する。そのあと、ステップＳ４０７に処理を戻し、新たな要素を選択する。

【００８２】ステップＳ４４２において非テキスト要素が線画や中間調画像であるのに十分な大きさであると判断されると、ステップＳ４４４に処理を進め、非テキスト要素が線画であるか中間調画像であるかの判別を行う。ステップＳ４４４からステップＳ４４８までの処理は、それぞれステップＳ４２８からステップＳ４３２までの処理と同一のものであり、ここでの説明は省略する。

【００８３】図４Ａ～図４Ｃ（図３中のステップＳ３０１）を参照して以上説明したように、画素画像中のすべての連結要素を検出して分類すると、図１４に示されるような木構造が得られる。図１４に示されるように、木構造の根は画素画像データのページに対応する。根からの子ノードには、テキストブロックや、未知，フレーム，絵，線などからなる非テキストブロックが存在する。また、フレームの子ノードには、テキストブロック，未知の非テキストブロック，テキストブロックを有する表，絵，線などが存在する。

【００８４】図１０は画素画像データの典型的なページ９０を示したものであり、大きなフォントサイズのテキスト９１，テキスト９３などを含む表９２，テキストデータ９４，水平線９５，もう一つのタイトル９６，２段落のテキストデータ９７及び見だし９９を備えるフレーム線画９８とタイトル１００で始まり，テキストデータ１０１，見だし１０３を備えるフレーム中間調画像１０２，テキストデータ１０４，水平線１０５，テキストデータの最終段落１０６へと続く第２段とが示されている。図１１は、ステップＳ３０１に基づく処理後の同一画素画像を示したものである。図１１に示されているように、画素画像データ９０中の連結要素が矩形ブロックで囲まれており、矩形ブロックの内部についてはステップＳ４１５からステップＳ４３４までのフレーム処理で判別される。

【００８５】ステップＳ３０２では、ステップＳ３０１で得られたすべてのテキスト要素を、木構造の位置に関わらず水平方向にグループ化する。グループ処理は各テキスト要素の密集性ならびにその近傍関係とに基づいて行われる。なお、この際、段に対応する垂直方向に伸びたギャップが検出され保持される。以下、図８を参照して、ステップＳ３０２で行う詳細な処理について説明を加える。図８に示される処理ステップは、ＲＯＭ１７に保持されるプログラムステップに基づいてＣＰＵ１０で実行される。

【００８６】ステップＳ８０１では、非テキスト要素の左端ならびに右端から垂直にギャップラインマーカを伸ばす処理を行う。すなわち、図１１に示されるように、ギャップラインマーカ１０９ａと１０９ｂとをテキストあるいは非テキスト要素（ここでは要素９５）に達するまで垂直に伸ばす。また、ギャップラインマーカ１０９ｃと１０９ｄとをテキストあるいは非テキスト要素（ここでは要素９５）に達するまで垂直方向に伸ばす。同様に、残りの非テキスト要素の左端ならびに右端からギャップラインマーカを垂直に伸ばす。このようなギャップラインマーカを用いることで、画素画像データにおいて段に対応するギャップ位置の判別が容易になる。

【００８７】ステップＳ８０２において、図１１のテキスト要素１０７のようなテキスト要素は、連結によりギャップラインマーカを横切らず、且つ他のテキスト要素と接しているかあるいは他のテキスト要素から所定のしきい値内の距離にあれば、１つのテキストラインに連結される。なお、適切なしきい値として、ステップＳ４０４で求めた平均テキスト長に経験的に得られたスカラー値を掛けたものを用いる（「１．２」を用いると良好な結果が得られる）。また、連結処理を行うに先立って、テキスト要素間の垂直ギャップを調べ、段構造を示唆するような垂直ギャップが存在するかどうかを判別する。すなわち、図１１に示されるように、ギャップ１０８はテキスト要素のペアの間に存在するが、このギャップはテキスト画像データ中で垂直方向に数ライン程度の長さとなるため、テキスト要素が他の要素から所定のしきい値内の距離に位置しているにも関わらず、ステップＳ８０２ではギャップが維持される。

【００８８】ステップＳ８０３では、連結ステップＳ８０２で連結されなかったテキスト要素のペアが連結によりギャップラインマーカを横切らず、隣のラインの第３テキスト要素と共に重なるような場合に、これらのテキスト要素のペアを連結する。このようなステップにより、段構造を示すギャップではなく単にテキストライン中のランダムな空間構成に起因するようなギャップを効果的に除去することが可能となる。例えば、図１１において、ステップＳ８０２ではギャップ１０８の連結は行われないが、ギャップ両側のテキスト要素は１行下のラ

インの第3テキスト要素と重なり、またギャップラインマーカを横切らないため、ステップＳ８０３においてこのギャップが除去される。

【００８９】ステップＳ８０４では、これらの処理結果に基づいて木構造の適切な更新が行われる。図１２は、ステップＳ３０２のグループ化処理の結果を示しており、図１５はステップＳ３０２のグループ化処理で修正された木構造を示している。図１２に示されるように、各々が接しているテキスト要素は、ライン１１０のようにテキストラインにグループ化される。すなわち、木構造中に位置するテキスト要素をテキストラインに連結する処理を行うが、１１１などのようにテキスト要素が木構造中のフレーム－表ノードの下に位置する場合にも連結処理が行われる。ここで、このようなグループ化処理は、上のステップＳ４１７からＳ４３９で求めた白輪郭境界を横切ることはなく、表中の個々の項目が1つの連続するテキストラインにグループ化されることはない。また、左段と右段との間のギャップは維持される。さらに、非テキスト要素は再連結されない。すなわち、１１２や１１３などの要素のように、互いに所定のしきい値内の距離にあっても、非テキスト要素のグループ化は行わない。

【００９０】図１５には、新たなグループ化処理の結果に基づいて修正した木構造が示されている。図８（図3中のステップＳ３０２）を参照して説明したようにテキスト要素のテキストラインのグループ化を行った後には、ステップＳ３０３に示されるようにテキストラインを垂直方向にグループ化してテキストブロックを生成する処理を行う。

【００９１】以下、図９を参照してこの処理について詳述する。グループ化処理は、テキストライン要素の密集性や非テキスト要素の位置に基づいて行われる。例えば、間に存在する非テキストラインは境界を示すため、これを利用すれば非テキストライン両側のテキストラインを1つのテキストブロックにグループ化することを避けられる。なお、処理は、2つの連続する非テキストライン要素間のすべてのテキストラインに対して1度に行われる。また、ステップＳ３０３では、テキスト要素を非テキスト要素に連結するべきか（例えば、非テキスト画像とそのテキスト見だし）、非テキスト要素を他の非テキスト要素に連結すべきか（例えば、中間調画像と線画）の判断も行う。

【００９２】図9は、テキストラインのテキストブロックへのグループ化を示す詳細なフローチャートである。ステップＳ９０１においては、最大の予想フォントサイズより小ではあったが平均テキストサイズより大であったため、ステップＳ４０４で非テキスト要素に分類された非テキスト要素からタイトルブロックの形成を行う。隣り合う非テキスト要素のうち同じようなサイズの要素に対してはすべてグループ化を行い、タイトルブロック

を形成し、「タイトル」属性をこのグループに付与する。ここでグループ化されなかった残りすべての非テキスト要素に対しては「絵－テキスト」属性が付与される。また、それに応じて木構造の更新も行う。ここで得られたタイトルは、ページ再生時（ステップＳ２１２）に有用な情報となる。

【００９３】ステップＳ９０２では、2つのテキストラインにまたがる非テキスト要素の位置を明確にする。このような非テキスト要素はテキストブロック間の境界となり、テキストラインを1つのテキストブロックにグループ化してしまうことを避けることができる。ステップＳ９０３では、2つのステップでテキストラインの垂直方向のグループ化を行い、テキストブロックを形成する。第1のステップでは、画素密度の垂直ヒストグラムを求めるなどして、段間のギャップを検出する。第2のステップでは、垂直方向に連続するテキストライン間の垂直距離がステップＳ４０４で求めたテキストの高さより小さければ、それぞれの段ごとにテキストラインのグループ化を行う。このステップＳ９０３の処理により、図2のライン１１４のような同一テキスト段落中のテキストラインを、テキストブロックにグループ化する。

【００９４】ステップＳ９０４では、垂直ならびに水平方向に隣接するテキストブロックのグループ化処理を行う。ここで、これらのテキストブロックが非テキスト要素で分離されていず、またこれらのブロックを連結してもステップＳ９０３のヒストグラムから求めたギャップが保持されるときに、テキストブロックのグループ化が行われる。また、ブロック間の距離がステップＳ４０４の垂直の高さに基づいて計算される所定のしきい値より小であるときに、テキストブロックはグループ化される。このステップＳ９０４の処理により、図１２中の段落１１５のラインと段落１１６のラインのテキストブロックはグループ化されるが、段落１１７と１１８のラインのテキストブロックはそれらの間に非テキスト要素１１９（線）を有するためグループ化が行われない。

【００９５】ステップＳ９０５では、テキストブロックを非テキストブロックに連結すべきか、非テキストブロックを他の非テキストブロックに連結すべきかを判断する。ここで、テキストブロックの非テキストータイトルブロック，非テキストー中間調ブロック，非テキストーライン接触ブロックとの連結は、以下のように行われる。

（1）テキストブロックが非テキストータイトルブロックと水平方向に近い位置にあり、垂直方向に重なっている場合には、テキストブロックを非テキストータイトルブロックに連結する。

（2）テキストブロックがワードサイズのブロックより小さくて（水平、垂直方向とも）、テキストブロックの隣にワードサイズのテキストブロックが存在しないときには、テキストブロックを非テキストー中間調画像ブロ

45

ックの中に位置付ける。

（3）テキストブロックが非テキストーライン接触ブロックと重なっていれば、ライン接触ブロックはテキストの下線である可能性が高いため、ライン接触ブロックをテキストブロックに変換する。

46

＊【0096】また、以下の表にしたがって、非テキストブロックは他の非テキストブロックと連結される。

【0097】

【表1】

	中間調	線画	テキストー絵	タイトル
中間調	テスト1	非連結	常に連結	非連結
テキストー絵	テスト1	テスト2	テスト2	テスト3
線画	テスト1	テスト1	非連結	非連結
タイトル	非連結	非連結	非連結	テスト3

この表中のテストは以下の通りである。

テスト1：1つのブロックが完全に他のブロック内に位置すれば連結、

テスト2：絵ーテキストの幅がワードサイズのブロックの幅より小さければ連結、

テスト3：ブロックが近接していれば連結。

【0098】ステップS906では、適切な属性が付与され、上述の処理結果に基づいて木構造の更新が行われる。図13は図9の処理を行った結果のブロック構造であり、図16はその木構造の例である。図13において、ブロックには、タイトルブロック120，テキストブロック121及び絵データ122が含まれる。また、フォームデータも含まれ、123は表構成のデータを有するフレーム要素を示しており、124はテキスト要素125を有するフレーム要素を示している。なお、非テキストライン画像127は図13に示されるさまざまな要素を分離している。

【0099】図3から図16を参照しながら説明したブロック選択処理を終えると、上述のように文字認識処理は図2のステップS204に処理を進める。すなわち、階層的木構造中の第1ブロックを選択して認識処理を行う。このブロックがテキストブロックでない場合には、処理をステップS204からステップS205に進め、木構造中の次のブロックを選択する。テキストブロックが選択されるまでステップS204とS205を繰り返し、テキストブロックが選択された時点でステップS2※06に処理を進め、ラインの分割処理を行う。

【0100】図17は、図2のライン分割ステップS206で実行される処理ステップを詳細に示すフローチャートである。図17に示される処理ステップは、プログラムROM17に保持されるコンピュータプログラムにしたがってCPU10で実行される。ステップS1701に先立って画像縮小処理を行っても良い。しかし、ライン分割処理や文字分割処理は水平方向の空白により影響を受けやすいため、画像縮小処理を行うにあたっては分割精度に影響を与えないような注意が必要である。すなわち、水平方向と垂直方向とでそれぞれ異なる画像縮小手法を用いることが好ましい。垂直方向では画素の結合を「OR」論理で行い、垂直方向の対象画素のうち1つでも黒画素が存在すれば黒画素が出力される。すなわち、垂直方向の2：1の画像縮小処理では、2つの垂直画素のいずれかが黒画素であれば黒画素が出力される。これに対して水平方向では画素の結合を「AND」論理で行い、水平方向の対象画素すべてが黒画素であれば黒画素が出力される。すなわち、水平方向の3：1の画像縮小処理では、3つの画素がすべて黒画素のときのみ黒画素が出力される。

【0101】垂直方向に3：1の縮小を行い、水平方向に2：1の縮小を行う場合の処理例を以下に示す（「0」は白画素を「X」は黒画素を表す）。

【0102】

【表2】

原画像	垂直縮小(OR)	水平縮小(AND)
X 0 X 0 X X X 0 0 0 0 0 0 X 0 0 0 X X X 0 0 X 0	X X X 0 0 X X X	X 0 0 X

画像縮小処理を終えると、この縮小画像に対してライン分割処理と文字切り出し処理とが行われる。ここで、図2の残りの認識処理（すなわち、ステップS208からステップS213まで）に影響を及ぼさないように、ライン分割処理と文字切り出し処理とを終えた時点で、文字間の切り出しは縮小していないもとの画像データに対して行う。

【0103】ステップS1701では、ステップS20

4で選択されたテキストデータブロックごとに画素密度の水平投影を求める。画素密度の水平投影は、画素画像の各行ごとに黒画素数を計数して得られる。ここで、画素密度の水平投影は全体のテキストブロックに対して求めることが好ましいが、これは本質的な点ではない。すなわち、テキストブロックを複数の画素列、例えば2あるいは3列に分割して、各列ごとに画素密度の水平投影を求めることができる。もちろん、このような処理では、本発明の処理時間短縮という利点は失われる。

【0104】ステップS1702では、水平投影のうちゼロでない領域を求め、どれかが最大フォントサイズと等しい所定のしきい値より大であるか否かを調べる。最大フォントサイズ以下であれば、水平投影はページ上のラインを均一に分割していることになり、処理をステップS1703に進める。ステップS1703では、画素密度の水平投影のうち近接している領域を連結する。この処理を図18A～図18Dを用いて説明する。図18Aは典型的なテキストブロック230を示しており、文字画像のライン231と233、ならびに雑音スポット232（すなわち、文字情報ではない黒画素）が含まれる。これに対応する画素密度の水平投影は234に示される。水平投影234に示されるように、領域235は文字「1」の上の点に対応し、領域236はライン231上の残りの文字に対応し、領域237と238は雑音スポット232に対応し、領域239はライン233上の文字に対応する。これらの各ピークにより、境界が水平投影のゼロ値となる領域が定義される。ステップS1703では、同一テキストライン上に存在するピーク235と236のような近接領域は連結され、テキストラインとは関係のないピーク237と238のような近接領域は連結されないことが望まれる。

【0105】ステップS1703で述べたように、近接領域を連結するためには、画素密度の水平投影を投影の上部から下部まで（テキストブロックの上部から下部まで）調べる。第1の領域を検出すると、その下の次の領域の水平投影をスキャンして、2つの領域の高さを比較する。上の領域の最も高い要素の高さが下の領域の高さより小さく、2つの領域の間隔が上の領域中の最も高い要素の高さより小さければ、これら2つの領域を連結する。ここで、この近接領域を連結する処理は、スケールに対して不変であることに注意されたい。すなわち、テキストの近接領域を連結する際には、テキストサイズが12ポイントであるとか8ポイントであるなどの知識は不要である。

【0106】そこで、図18Aに戻り、領域235の高さと領域236の高さとの比較を行うと、上の領域235の高さが下の領域の高さより小さいことがわかる。また、2つの領域間のギャップは領域235の高さより小さいと判断される。したがって、図18Bに示されるように、領域235と236とは1つの領域236'として連結される。

【0107】画素密度の水平投影を下方向に調べると、領域237が検出される。この場合、領域236'の高さが領域237の高さ以上であるため、これら2つの領域は連結されない。さらに、画素密度の水平投影を下方向に調べると、領域237の高さが領域238の高さ以下であり、また領域間のギャップは領域237の高さ以下であることがわかる。そこで、図18Cに示されるように、領域237と238とは1つの領域238'として連結される。画素密度の水平投影をさらに下方向に調べると、領域239が検出される。この場合、領域238'の高さは領域239の高さより低いが、領域間のギャップは領域238'を構成している領域237と238とのどちらかの高さより大きいことがわかる。そこで、これら2つの領域は連結されない。

【0108】また、近接領域の連結処理のあとに各領域の高さを調べて各領域の高さがラインの最小の高さに対応するしきい値より大であるかどうかを判別することもできる。この際、しきい値は、これまでに検出された領域の高さの平均として適当に設定される。領域がしきい値以下の高さの場合には、領域を画素データ中の雑音スポットに起因するものと判断し削除することができる。したがって、領域238'の高さは、領域236'，238'，239の高さの平均として決められるしきい値以下であるため、図18Dに示されるように領域238'は削除される。

【0109】これらの処理を終えると、図17に戻って処理をステップS1704に進め、領域を個々のラインセグメントに分割し、図2のステップ207で示される文字切り出しに処理を進める。ステップS1702において、ステップS1701で処理された領域が大きすぎる領域であった場合には、テキストラインが傾いていると判断される。例えば、図19Aに示されているように、テキストブロック240には複数の傾いたテキストライン241が含まれる。ステップS1702に基づく処理では、244に代表テキストを示したようにテキストラインが水平方向に相互に重なりあってしまうため、242のような画素密度の水平投影が得られる。

【0110】そこで、処理をステップS1705に進め、テキストブロックを段に分割する。図19Bに示されるように、テキストブロックの段数を2倍にする。すなわち、テキストブロック240を2つの段に分割する。この際、少なくとも1つの共通の画素が重なり合うように段を分割することが好ましい。また、テキストブロックを2ブロック以上、例えば3あるいは4ブロックに分割することも可能である。

【0111】ステップS1706では図19Bの247や249のように各段ごとに画素密度の水平投影を求め、ステップS1707で領域が大きすぎるかどうかを再び調べる。領域が大きすぎる場合には、処理をステッ

プＳ１７０８に進め、段数を再び増加させる。例えば、図１９Ｃに示されているように、段数を更に２倍にする。また、ステップＳ１７０９において、段の幅が最低限度より大であるかどうかを確認する。この最低限度は、これ以上段数を増加させると適切なライン分割が不可能になる点を示している。好適な実施例では、最低限度は１６画素幅である。ステップＳ１７０９において最低限度に達していると判断されると、ライン分割は不可能であると表示して処理を終了する。一方、最低限度まで達していない場合には、処理をステップＳ１７０６に戻し、新たな段ごとに水平投影を再び計算する。ステップＳ１７０７において領域が大きすぎないと判断された場合には、処理をステップＳ１７１０に進める。すなわち、図１９Ｃに示されるように、ラインセグメント以下の幅の領域が割り出された。そこで、処理をステップＳ１７１０に進め、ステップＳ１７０３で説明したように近接領域を連結する。そして、ステップＳ１７１１において、各段ごとに単一ラインセグメントに対応する領域を割り出す。

【０１１２】すなわち、図１９Ｃにおいて、単一ラインセグメントに対応する領域２５０，２５１，２５２，２５３を検出する。そして、領域が異なる段間で接触しており単一ラインセグメントを構成するかどうかを判別するために、各段を上から下に調べ各段の第１領域を検出する。そこで、その領域に接触している領域を調べ、図１９Ｄに示されているように２つの距離を求める。この２つの距離、（１）２つの領域を合せたときの全長距離Ａと、（２）２つの領域の共有領域すなわち２つの領域の交差領域の距離Ｂとの２つである。そして、比Ａ／Ｂを求め、２つの領域が多くの部分で重なっているかを確かめるためにしきい値と比較する（しきい値として５を用いると良好な結果が得られる）。比Ａ／Ｂがしきい値より小であれば、２つのブロックは多くの部分で重複していることになり、ブロックは単一ラインセグメントを構成していると考えられる。そこで、ステップＳ１７１２において、比Ａ／Ｂで求められる接触領域を単一ラインセグメントとして割り出される。

【０１１３】ここで、比Ａ／Ｂの計算と、比Ａ／Ｂとしきい値との比較処理とは、スケールに不変な処理であり、ライン中のテキストサイズに関わらず、重なった接触領域は単一ラインセグメントとして割り出される。このようなスケール不変の性質は、ラインセグメント中のテキストサイズが既知である必要がないという点で望ましいものである。

【０１１４】図１７で述べたラインの分割処理を終えると、図２のステップＳ２０７で説明し、詳細が図２０に示される文字分割（文字切り出し）に処理を進める。図２０に示されるように、文字切り出しは多階層の処理で行われ、各階層は徐々に複雑な文字切り出し処理を行う。すなわち、文字切り出し処理は３つの処理に分類される。相互に接触ならびに重なっていない文字間の切り出し、相互に接触してはいないが重なっている文字間の切り出し、接触している文字間の切り出しの３つである。例えば、図３４Ｂに示されるように、文字「Ｓ」と「ａ」は接触ならびに重なっていないため第１の処理に分類される。一方、文字「ｆ」と「ｙ」は接触してはいないが重なっているため第２の処理に分類される。さらに、文字「ｔ」と「ｌ」は相互に接触しているため第３の処理に分類される。

【０１１５】図２０に示されるように、各階層は３つの処理のうちの１つを実行するように構成される。すなわち、階層１（２６１）では接触ならびに重なっていない文字間の切り出しが行われる。階層１に続いて、当該テキストの性質及び特徴に関する知識の有無に応じて処理を進める。テキストが単一スペースのテキストであれば、すなわち文字が垂直で文字間が均一であれば（例えば「クーリエ」フォント）、処理を階層２（２６２）に進め、接触している文字の切り出しを行う。単一スペースの文字であっても、コピーやファクシミリ転送の繰り返しに起因する画像の劣化のため、文字が相互に接触することがある。そして、文字認識処理２６３に処理を進め、図２のステップＳ２０９に進む。

【０１１６】一方、テキストブロックの性質及び特徴に関して何の情報もない場合、あるいはテキストが単一スペースのテキストでなければ、処理を階層２（２６４）に進め、接触はしていないが重なっている文字間の切り出しを行う。そして、階層１と階層２との切り出しで得られた文字に対して２６５で認識処理を行う。認識処理２６５で認識されなかったすべての文字に対しては、階層３（２６６）の処理を行う。すなわち、文字が認識されない理由として、文字の切り出しが完全に行われないため実際には認識不能な文字は接触している複数の文字と考えられる。そこで、階層３では接触している文字の切り出しが行われる。階層３で文字を切り出すと、２６７で文字認識処理を行う。認識処理が成功した場合には、処理を図２のステップＳ２０９に戻す。一方、認識処理が再び失敗した場合には、階層３での切り出しが不適切であったと判断する。そこで、認識不能であった切り出しを再接続し、再び階層３の切り出し処理と認識処理とを行う。このような処理を、文字の切り出しが不可能になるまで繰り返し行う。

【０１１７】「単一スペース（mono-spaced）」２６２か「全スペース(all-spacing)」２６４かのどちらに処理を進めるかは、オペレータからの入力によって選択される。オペレータからの入力がない場合には、デフォルトとして処理を「全スペース」２６４に進める。というのは、この階層は単一スペースのみならず非単一スペース文字にも適用できるためである。

【０１１８】図２１から２８までは階層１から階層３までの処理を説明する図であり、図３０と図３１は２６９

で示される再接続処理を説明する図である。図２１は非接触で重なっていない文字を切り出す階層１の切り出し処理を説明するフローチャートである。階層１では、２つの文字間の白画素すなわち空白を検出することにより、非接触で重なっていない文字の切り出しを行う。

【０１１９】具体的には、図２１のステップＳ２１０１に示されるように、空白でない画素すなわち黒画素が検出されるまでラインセグメント中をとびとびに検索し、文字間の白スペースを検出する。ここで、とびとばの検索とは、ラインセグメントのすべての画素を検索するのではなく、図２２に示されるように、ラインセグメントの１つの段中の一部の画素２７１のみを検索するものである。なお、ラインセグメントの段中の全画素の１／３のみ、すなわち３画素ごとの検索で十分であるとの結果が得られている。画素２７１中で空白でない画素すなわち黒画素が検出されなかった場合には、２７２で示すように数列の画素をとばして、例えば３画素ごとに新たな列で黒画素を検索する。ここで、画素を３列とばしても、非接触で重なっていない文字の検出性能は低下しないとの結果が得られている。このようなとびとびの検索を、図２２の画素２７４のような最初の黒画素が検出されるまで繰り返す。

【０１２０】最初の黒画素が検出されるとステップＳ２１０２に処理を進め、ラインセグメントを後向きに検索し、完全に空白な列を検出する。ここでの検索は、ステップＳ２１０１の検索とは異なり、各列ごとにすべての画素を検索して完全に空白な列を検出するものである。すなわち、図２２に示されるように、完全に空白な列２７６が検出されるまで後向きステップ２７５が実行される。

【０１２１】完全に空白な列が検出されると、処理をステップＳ２１０３に進め、完全に空白な列が検出されるまで画素２７４の列から前向きに検索される。ステップＳ２１０２と同様に、ここでの前向き検索も各列ごとにすべての画素を検索するもので、２７７で示すように前向きに検索される。図２２の２７８で示されるような完全に空白な列が検出されるまで前向き検索が行われる。

【０１２２】ステップＳ２１０３で完全に空白な列が検出されると、処理をステップＳ２１０４に進め、空白列２７６と２７８とで文字を切り出す。その後、処理をステップＳ２１０１に戻し、空白でないすなわち黒画素が再び検出されるまで、ラインセグメントでのとびとびの検索を再び行う。ラインセグメント全体で階層１の処理を終えると、テキストが単一スペース（クーリエフォントなど）であるか、あるいはテキストのスペースが未知もしくは単一スペース以外（比例フォントなど）であるかに応じて、図２０の２６１あるいは２６４の階層２の処理に処理を進める。テキストが単一スペースであれば、２６１の階層２の切り出しに処理を進める。

【０１２３】図２３は単一スペーステキストのための階層２の処理の流れを示すフローチャートである。なお、図２３に示される処理ステップは、ＲＯＭ１７に保持されるプログラムステップに基づいてＣＰＵ１０で実行される。階層２の処理に先立って、文字セグメントの幅を調べて、過小サイズの文字セグメントを割り出す。ここで、文字セグメントの幅が平均文字幅の半分以下のときに過小サイズであると判断する。過小サイズの文字セグメントが隣り合って存在する場合には、階層１の処理で１つの文字を半分ずつ２つに切り出してしまった可能性が高いため、この過小サイズ文字のペアを連結する。

【０１２４】ステップＳ２３０１では、各文字ブロックの幅をすべての文字ブロックの平均幅と比較して、階層１で切り出された文字ブロックのうち過大サイズのものを割り出す。ここで、各文字ブロックの幅とすべての文字ブロックの平均幅との比較は、文字が単一スペースであって各文字がほぼ同一の幅を有することが既知であるため、過大サイズの文字ブロックの検出には有効な処理となる。文字ブロックの幅（「W」）が以下の式を満たすと、文字ブロックが過大サイズであると判断される。

【０１２５】

【数５】$W > (1+c) * W_{ave}$

ここで、ｃは定数、W_{ave}はラインセグメント中のすべての文字ブロックの平均幅である。なお、この判別処理はスケールに不変であることに注意されたい。定数ｃは、以下のように単一スペースフォントの統計的性質に基づいて決定される。クーリエアルファベットなどの単一スペースアルファベットの各文字は、単一なスペースを有しており、各スペースは文字が存在する部分α_iと文字の回りの空白スペースの部分β_iとからなる。

【０１２６】例えば、図２４の文字「ｅ」に示されるように、文字「ｅ」が存在するスペースは中心領域α_iと周りの空白スペースβ_iとからなる。ここで、ｉは「ｅ」に対応する番号であり、すべてのｉに対して$\alpha_i + \beta_i = 1$である。α_iとβ_iについてはアルファベット中の各文字、すなわち英字、数字、記号などごとに求めることができ、α_iとβ_iの平均値（それぞれαとβ）ならびに標準偏差（それぞれσ_αとσ_β）を計算できる。そこで、定数ｃは以下の式で求める。

【０１２７】

【数６】$c = \sigma_\beta / \alpha$

図１の装置で用いるクーリエ文字セットでは、$\alpha = 25/35$，$\sigma_\beta = 10/35$となるためｃ＝０．４となる。ステップＳ２３０１で過大サイズの文字ブロックが割り出されると、ステップＳ２３０２に処理を進め、過大サイズブロック中に含まれるおおよその文字数を算出し、おおよその文字境界を求める。具体的には、図２４において、ブロック２８０の幅Ｗは、すべてのブロック２８０から２８３の平均の幅に（１＋ｃ）を乗算して計算されるしきい値より大であるため、ブロック２８０は過大サイズの文字ブロックと判断される。そして、幅Ｗ

をαで割った値を最も近い整数値に丸めて、過大サイズブロック２８０中のおおよその文字数を算出する。

【０１２８】

【数7】文字数N＝〔W／α〕（最も近い整数）

また、ここで得られたブロック中のおおよその文字数「N」に基づいて、過大サイズのブロックを単一に分割して、おおよその文字境界を求める。ステップＳ２３０３では、ブロック中の画素の垂直投影特性２８４を、おおよその文字境界の近傍２８５で求める。ここで、垂直投影特性２８４を求める近傍は、距離σ₁に基づいて決められる。すなわち、図２４に示されるように、おおよその文字境界の近傍±σ₁で垂直投影特性２８４を求める。

【０１２９】ステップＳ２３０４では、各垂直投影特性２８４中での最小位置２８６を割り出し、この最小位置２８４で文字の切り出しを行う。図２０の２６１の階層２の処理を終えると、文字認識２６３、そしてさらに図２のステップＳ２０９に処理を進める。ラインセグメント中の文字のスペースが未知あるいは単一スペースでない場合には、文字がラインセグメント中で単一のスペースを有しているとは限らない。そこで、図２０の２６４の階層２の処理に進み、非接触であるが重なっている文字間の切り出しを行う。図２５と図２６とはこの処理を説明する図である。

【０１３０】ステップＳ２５０１において、階層１で切り出された各文字を分析して、文字ブロック中の各画像の輪郭のアウトラインを追跡する処理を行う。すなわち、図２６Ａに示されるように、文字「ｆｙ」を有する文字ブロックは非接触だが重なっている文字「f」と「y」を含み、これらは重なっているため階層１の処理では切り出されない。そこで、図２６Ｂに示されるように、まず、この文字ブロックをブロックの右下から左方向ならびに上方向に調べて黒画素を検出する。黒画素を検出すると、図２６Ｃの２８７のように黒画素と黒画素とを接続したものである輪郭の追跡が行われる。第１の文字に対して全体の輪郭追跡を終えると、２８８のように文字ブロック中のすべての黒画素の輪郭を追跡するまで、スキャン処理を続ける。このような処理により、各々が分離した非接触の文字が得られ、図２６Ｄに示されるようにこれらの文字が文字ブロックから切り出される。

【０１３１】階層２の処理では、非接触だが重なっている文字の切り出しとともに、複数のストロークからなる単一文字、例えば「i」，「j」，「:」，「;」，「!」，「=」，「%」をも分離してしまう。そこで、ステップＳ２５０２でこのような文字の再接続を行う。図２７はこの処理の詳細なフローチャートである。図２７に示す再接続処理の対象となるのは階層２の処理で切り出された文字のみであり、特定の条件を満たすときのみ再接続が行われる。具体的には、ブロックが重なる、すなわち左側文字の最も右側の画素が右側文字の最も左側の画素の上あるいは下に位置するような場合にのみ、ブロックの再接続が行われる。

【０１３２】そこで、ステップＳ２７０１でブロックが重なっているかどうかを調べる。ブロックが重複していなければ、再接続処理は不必要であり（ステップＳ２７０２）、再接続処理を終了する。一方、ブロックが重複していれば、ステップＳ２７０３に処理を進め、ブロックが垂直方向に分離しているかどうかを調べる。ここで、ブロックが垂直方向に分離していると、「i」，「j」，「:」，「;」，「!」，「=」などの複数ストローク文字が階層２の処理で切り出されてしまった可能性があるため、これらの文字であるかどうかを調べる。

【０１３３】これらのブロックは垂直方向に分離されているため、高さがＨ１の上部ブロックと高さがＨ２の下部ブロックとを含む。これらの高さの計算はステップＳ２７０４で行われ、ステップＳ２７０５でＨ２が（２×Ｈ１）より大であれば文字「i」あるいは「j」が分離したものである可能性が高い。そこで、文字の重複度を計算する（ステップＳ２７０６）。すなわち、２つの部分に隣接する最も右側の４画素列を平均して、この平均位置の差を計算する。（全体位置ではなく最も右側の位置を平均したのは、文字「i」や「j」の上のドットが「i」や「j」の中心ではなく上部のセリフの右側に位置するためである）。この際、スケール不変の処理とするために、平均位置の差が本体のうちの小さい幅に定数を乗じたものより小であれば、ブロックを再接続するものとする。ここで、定数は、分離の程度が予測できないような劣化画像でも再接続できるように選択され、本実施例では「９／８」としている。

【０１３４】一方、ステップＳ２７０５でＨ２が（２×Ｈ１）以下、すなわち下部ブロックの高さが上部本体の２倍以下である場合には、「:」，「;」，「!」，「=」などの文字が切り出された可能性が高い。そこで、ステップＳ２７０７に処理を進め、この可能性を調べる。すなわち、各本体中の隣接する４画素列の平均中心値を求め、これらの中心値の差を得る。この際、スケール不変の処理とするために、中心値の差が２つの本体の幅のうちの小さい方に定数を乗じたものより小であれば、上述の文字の１つである可能性が高いためブロックを再接続するものとする。なお、上述のように、定数として「９／８」を用いると良好な結果が得られている。

【０１３５】ステップＳ２７０３でブロックが垂直方向に分離されない場合（すなわち、２つのブロック間に水平方向に伸びるギャップが存在しない場合）には、文字はタイプ１のパーセント記号（「タイプ1」）、タイプ２のパーセント記号（「タイプ2」）、タイプ３のパーセント記号（「タイプ3」）のどれかである可能性が高い。そこで、ステップＳ２７０８で以下のように順々に

55

各タイプごとにチェックする。なお、変数は以下の通りである。

【0136】W1：第1文字の幅（左から右）
W2：第2文字の幅
H1：第1文字の高さ（上から下）
H2：第2文字の高さ
L1：第1文字の左端画素の列
R1：第1文字の右端画素の列＋1画素
L2：第2文字の左端画素の列
R2：第2文字の右端画素の列＋1画素
注意：L1は常にL2以下である。

【0137】まず、タイプ1のパーセント記号をチェックする。以下の2つの条件が満たされるときにタイプ1のパーセント記号であると判断され、ブロックが連結される。

1）0．24＜min（W1，W2）／max（W1，W2）＜0．77
これはドット幅とライン幅との比の条件である。

【0138】2）［min（R1，R2）－max（L1，L2）］／min（W1，W2）＞0．76
これはブロックが水平方向に大部分重なっているという条件である。次いで、タイプ2のパーセント記号をチェックする。以下の4つの条件が満たされるときにタイプ2のパーセント記号であると判断され、ブロックが連結される。

【0139】1）（0．25）L2 ＜ R1－L2
これはブロックが水平方向に十分重なっているという条件である。

2）0．50 ＜ W1／W2 ＜ 1．10
これはドット幅とライン幅の適切な比率の条件である。

3）0．43 ＜ （H1／H2） ＜ 0．70
これはドット高とライン高の適切な比率の条件である。

【0140】4）（1／m） ＞ 0．37
ここで、mはパーセント記号の「斜線」部上のP1とP2とを結ぶ線の傾きである。なお、P1とP2とは以下の手法で求められる。

P1：P1は第2文字の上部からD行目の行で、第2文字のプリントテキストを含む左端の画素の位置である。
ここで、変数DはD＝（0．1）W2である。

【0141】P2：P2は第2文字の下部からD行目の行で、第2文字のプリントテキストを含む左端の画素の位置である。

さらに、タイプ3のパーセント記号をチェックする。以下の条件が満たされるときにタイプ3のパーセント記号であると判断され、ブロックが連結される。

1）（0．25）L1 ＜ R2－L1
これはブロックが水平方向に十分重なっているという条件である。

【0142】2）0．50 ＜ W2／W1 ＜ 1．10

56

これはドット幅とライン幅の適切な比率の条件である。

3）0．43 ＜ （H2／H1） ＜ 0．70
これはドット高とライン高の適切な比率の条件である。

4）（1／m） ＞ 0．37
ここで、mはパーセント記号の「斜線」部位上のP1とP2とを結ぶ線の傾きである。なお、P1とP2とは以下の手法で求められる。

【0143】P1：P1は第1文字の上部からD行目の行で、第2文字のプリントテキストを含む右端の画素の位置である。ここで、変数DはD＝（0．1）W2である。

P2：P2は第1文字の下部からD行目の行で、第2文字のプリントテキストを含む右端の画素の位置である。

図20の264で説明した（また、図23から図27で詳細に説明した）階層2の切り出し処理と再接続処理を終えると、265の認識処理を切り出された文字に対して行う。階層1と階層2の切り出しにおいてラインセグメント中のほとんどの文字は適切に切り出されているため、265の認識処理では階層1と階層2で切り出された文字のほとんどを認識することができる。これに対し、265で認識不能であった文字は、その文字ブロックに接触文字が含まれている可能性が高い。そこで、このような認識不能の文字ブロックに対して、266の階層3の切り出し処理を行い、接触文字を切り出す。

【0144】図28は階層3の切り出し処理を示すフローチャートであり、図29Aから図29Dは接触文字を階層3で切り出す処理を説明する図である。図28に示される処理ステップはROM17に保持されており、CPU10で実行される。一般に階層3の切り出し処理は、文字ブロックを斜めに切り出して行われる。斜めの切り出し線の傾きと位置とは、ブロック中の画素密度の垂直投影特性を求め、垂直投影特性中で最も深い谷の側面の傾きを求めることで算出される。そこで、再び画素密度の垂直方向以外の投影を行う。すなわち、垂直投影特性中の谷側面の傾きに対応する回転角度方向に画素密度の投影を行う。こうして得られた複数の密度投影の中での最小点を検出し、最小点が得られた角度と位置で切り出しを行う。以下、この処理について詳細な説明を行う。

【0145】ステップS2801において、画素密度の垂直投影特性を求める。例えば、図29Aに示されるように、接触文字「t1」に対して垂直投影特性を求める。ステップS2802では、垂直投影特性中の第1の谷を検出する。垂直投影特性はデジタルであるため（すなわち、離散的な画素数の和であるため）、滑らかではなく、谷は垂直投影特性中の最小値が第1の低しきい値より下であって、第2の高いしきい値より大きい極大点で両側が囲まれた点であることで見付け出される。したがって、図29Aに示されるように、垂直投影特性を調べて、高いしきい値292より高い上の点で囲まれてい

るような低しきい値２９１より下の点が存在するかどうかを判別する。このような条件を満たす点が検出されれば、処理をステップＳ２８０３に進める。このような条件を満たす点が検出されなければ、以下に述べるようにしきい値の変更を行う。

【０１４６】まず、低いしきい値２９１を垂直投影特性の最大値の１０％とし、高いしきい値２９２を垂直投影特性の最大値の２０％とする。ここで、高いしきい値と低いしきい値に関する条件を満たす点が検出されない場合には、高いしきい値と低いしきい値との双方とも垂直投影特性の最大値の２％だけ増加させる。図２９Ａでは、しきい値２９１と２９２の条件を満たす点は検出されない。そこで、しきい値を図２９Ｂに示すように増加させて、低いしきい値２９１以下であって、高いしきい値２９２より高いの点２９５と２９６が両側に存在するような点２９４を検出する。点２９４を検出すると、ステップＳ２８０３に処理を進め、点２９４を囲む谷側面の傾きを算出する。谷の右側面の傾きは点２９４と２９５とを結ぶ線の傾きでありθ_1で示される。同様に、谷の左側面の傾きは点２９４と２９６とを結ぶ線の傾きでありθ_2で示される。

【０１４７】そこで、ステップＳ２８０４に処理を進め、角度θ_1とθ_2および角度θ_1とθ_2の近傍で回転させて投影特性を求める。すなわち、回転投影特性を角度θ_1，$\theta_1 \pm 3°$，$\theta_1 \pm 6°$，θ_2，$\theta_2 \pm 3°$，$\theta_2 \pm 6°$で求める。この回転投影特性は文字ブロック中の画素を三角関数変換することで求められる。より簡易な手法として、各回転角度（最も近い角度に近似）ごとに画素位置が示される表を用意して、テーブルルックアップで回転投影特性を求めることもできる。なお、回転投影特性の各点は、この画素位置の和として求められる。

【０１４８】図２９Ｃの２９７は回転投影特性の典型例を示したものである。回転投影特性２９７上の各点は、ここではθ_1の回転方向の画素数の和として求められる。上述のように、ここでの和は、文字ブロック中の画像を三角関数変換することで求めることもできるし、各回転角度ごとに用意された表を参照して簡易に求めることもできる。

【０１４９】図２９ＣとＤの点線で示されるようなすべての回転投影特性を求めると、ステップＳ２８０５に処理を進め、各回転投影特性（１０個すべて）ならびにステップＳ２８０１で求めた垂直投影特性とを比較して、すべての投影特性の中で最小点を検出する。最小点となる投影特性の角度が切り出し角度に対応する。すなわち、図２９Ｃに示されるように、点２９９が１１個の投影特性の中で最小点となれば、文字ブロックの切り出しが角度θ_1で最小点２９９の位置で行われる（ステップＳ２８０６）。

【０１５０】階層３の切り出し処理を終えると、切り出された文字に対して認識処理２６７を行う。ここで、階層３で切り出された２つの文字ブロックとも認識が行われたならば、図２のステップＳ２０９に処理を進める。しかし、ここでもなお認識不能なブロックが残る可能性があり、不適切な切り出しが行われた可能性と切り出された文字ブロックを再接続すべきであるという点を考慮する必要がある。この処理は再接続ブロック２６９で行われ、図３０に詳細に示されている。

【０１５１】ステップＳ３００１では、階層３での切り出し部分の双方に対して２６７の認識処理を試みる。ステップＳ３００２で２つの要素の認識が可能であると判断されると、上述の図２のステップＳ２０９に処理を進める。一方、２つの切り出し要素とも認識不能である場合には、ステップＳ３００３において少なくとも１つの要素が認識可能であるかどうかを判別する。２つの要素とも認識不能である場合にはステップＳ３００４に進み、各要素に対して階層３の切り出し処理とステップＳ３００１などの処理とを行う。

【０１５２】一方、ステップＳ３００３において少なくとも１つの要素が認識可能であると判断されると、ステップＳ３００５に処理を進め、認識不能の要素に対してさらに階層３の切り出し処理を行う。そして、ステップＳ３００６で新たに切り出された要素が認識可能であると判断されれば、認識不能な要素はなくなり図２のステップＳ２０９に処理を進める。これに対して、新たに切り出された要素がともに認識不能であるとステップＳ３００６で判断されると、ステップＳ３００７に処理を進め、不適切な切り出しブロックを再接続することを考える。

【０１５３】図３１は、階層３の処理において不適切な切り出しが行われる可能性を説明する図である。図３１Ａはイタリック文字「ｈｍ」がかなり劣化している例を示したものである。図２０の２６６の階層３の処理では第１の切り出しを３０１で行うため、文字「ｈ」の垂直部とループ部とが分離してしまう。そして、切り出された各要素に対して２６７で認識処理を行い、第１のブロックは「ｌ」として認識され、第２のブロックは認識不能と判断されたとする。

【０１５４】このような場合、処理はステップＳ３００５に進み、認識不能の要素３０３に対して図３１Ｂのようにさらなる階層３の切り出し処理が行われる。階層３の処理により、３０４でさらなる切り出しが行われ、切り出された要素３０５と３０６に対して２６７の認識処理が行われる。しかし、要素３０５は文字として認識不能であるため、再接続処理が必要であると判断される。

【０１５５】そこで、ステップＳ３００７において、認識不能の切り出し要素を、あらかじめ切り出された隣接する要素と接続する。この際、隣接する要素は認識されたものでも、認識不能のものでも良い。すると、図３１Ｃのように、要素３０２を要素３０５と再接続して文字

59

「h」をほとんど含む新たな要素３０２'が生成される。そこで、要素３０２'と要素３０６とに対して２６７の認識処理を行う（ステップＳ３００８）。

【０１５６】これらの処理を終えると、ステップＳ３００１に処理を進め、２つの要素ともに認識されたかどうかを調べる。この例の場合には、２つの要素とも文字「h」、「m」として認識されたため処理を終了する。一方、２つの要素とも認識不能であった場合には、上述の処理を繰り返す。

【０１５７】

【発明の効果】以上説明したように、本発明により、高速かつ正確に文書上の文字を認識してテキストファイルを作成する文字認識方法及び装置を提供できる。また、高速かつ正確に文書上のテキストと非テキストとを選別して、テキストブロックを割り出す方法及び装置を提供できる。

【０１５８】また、高速かつ正確に文書上の非テキストを分類できる方法及び装置を提供できる。また、高速かつ正確にテキストブロックからテキストラインを分割できる方法及び装置を提供できる。また、高速かつ正確に傾いた文書のテキストブロックからテキストラインを分割できる方法及び装置を提供できる。

【０１５９】また、高速かつ正確にテキストラインから文字を切り出す方法及び装置を提供できる。また、不適切に切り出された文字の再切り出しが可能な方法及び装置を提供できる。

【図面の簡単な説明】

【図１】本実施例の装置のブロック図である。

【図２】文字認識処理の流れを示すフローチャートである。

【図３】本実施例に基づくブロック分類と選別の処理の流れを示すフローチャートである。

【図４Ａ】画素画像データ中の連結部位の分類処理の流れを示すフローチャートである。

【図４Ｂ】画素画像データ中の連結部位の分類処理の流れを示すフローチャートである。

【図４Ｃ】画素画像データ中の連結部位の分類処理の流れを示すフローチャートである。

【図５Ａ】輪郭追跡を説明するための図である。

【図５Ｂ】輪郭追跡を説明するための図である。

【図５Ｃ】輪郭追跡を説明するための図である。

【図６Ａ】非テキスト部位の分類処理を説明するための図である。

【図６Ｂ】非テキスト部位の分類処理を説明するための図である。

【図６Ｃ】非テキスト部位の分類処理を説明するための図である。

【図７Ａ】白輪郭処理を説明するための図である。

【図７Ｂ】白輪郭処理を説明するための図である。

【図７Ｃ】白輪郭処理を説明するための図である。

60

【図７Ｄ】白輪郭処理を説明するための図である。

【図８】他のテキスト部位のサイズと近さに基づいてテキスト部分を水平方向に選択的に連結してテキストラインを形成する処理の流れを示すフローチャートである。

【図９】他のテキストラインのサイズと近さに基づいてテキストラインを垂直方向に選択的に連結してテキストブロックを形成する処理の流れを示すフローチャートである。

【図１０】代表的な画像の画素データを示す図である。

【図１１】ブロック分類と選別処理を説明するための図である。

【図１２】ブロック分類と選別処理を説明するための図である。

【図１３】ブロック分類と選別処理を説明するための図である。

【図１４】図１１から１３にそれぞれ対応する典型的な階層的木構造である。

【図１５】図１１から１３にそれぞれ対応する典型的な階層的木構造である。

【図１６】図１１から１３にそれぞれ対応する典型的な階層的木構造である。

【図１７】本実施例のライン分割処理の流れを示すフローチャートである。

【図１８Ａ】本実施例のライン分割処理を説明するための図である。

【図１８Ｂ】本実施例のライン分割処理を説明するための図である。

【図１８Ｃ】本実施例のライン分割処理を説明するための図である。

【図１８Ｄ】本実施例のライン分割処理を説明するための図である。

【図１９Ａ】本実施例のライン分割処理を説明するための図である。

【図１９Ｂ】本実施例のライン分割処理を説明するための図である。

【図１９Ｃ】本実施例のライン分割処理を説明するための図である。

【図１９Ｄ】本実施例のライン分割処理を説明するための図である。

【図２０】本実施例の文字切り出し処理の機能ブロック図である。

【図２１】図２０の階層１の文字切り出し処理の流れを示すフローチャートである。

【図２２】階層１の切り出し処理を説明するための図である。

【図２３】図２０における単一スペースモード（クーリエフォントなど）の階層２の文字切り出し処理の流れを示すフローチャートである。

【図２４】階層２の文字切り出し処理を説明するための図である。

61

【図２５】図２０における全スペースモード（比例スペースなど）の階層２の文字切り出し処理の流れを示すフローチャートである。
【図２６Ａ】階層２の文字切り出し処理を説明するための図である。
【図２６Ｂ】階層２の文字切り出し処理を説明するための図である。
【図２６Ｃ】階層２の文字切り出し処理を説明するための図である。
【図２６Ｄ】階層２の文字切り出し処理を説明するための図である。
【図２７】階層２の処理で切り出された複数ストローク文字を再融合するための再融合手法を示すフローチャートである。
【図２８】図２０の階層３の処理の流れを示すフローチャートである。
【図２９Ａ】階層３の文字切り出し処理を説明するための図である。
【図２９Ｂ】階層３の文字切り出し処理を説明するための図である。
【図２９Ｃ】階層３の文字切り出し処理を説明するための図である。
【図２９Ｄ】階層３の文字切り出し処理を説明するための図である。
【図３０】図２０の階層３の処理で切り出された部位の再融合処理の流れを示すフローチャートである。
【図３１Ａ】再連結処理を説明するための図である。
【図３１Ｂ】再連結処理を説明するための図である。
【図３１Ｃ】再連結処理を説明するための図である。
【図３２】文字認識すべき文書のページの代表例である。
【図３３Ａ】従来のライン分割手法を説明するための図である。
【図３３Ｂ】従来のライン分割手法を説明するための図である。
【図３３Ｃ】従来のライン分割手法を説明するための図である。
【図３４Ａ】従来の文字切り出し手法を説明するための図である。
【図３４Ｂ】従来の文字切り出し手法を説明するための図である。
【図３５】本実施例のブロック選別プログラムのソースコードを示す図である。
【図３６】本実施例のブロック選別プログラムのソースコードを示す図である。
【図３７】本実施例のブロック選別プログラムのソースコードを示す図である。
【図３８】本実施例のブロック選別プログラムのソースコードを示す図である。
【図３９】本実施例のブロック選別プログラムのソース

62

コードを示す図である。
【図４０】本実施例のブロック選別プログラムのソースコードを示す図である。
【図４１】本実施例のブロック選別プログラムのソースコードを示す図である。
【図４２】本実施例のブロック選別プログラムのソースコードを示す図である。
【図４３】本実施例のブロック選別プログラムのソースコードを示す図である。
【図４４】本実施例のブロック選別プログラムのソースコードを示す図である。
【図４５】本実施例のブロック選別プログラムのソースコードを示す図である。
【図４６】本実施例のブロック選別プログラムのソースコードを示す図である。
【図４７】本実施例のブロック選別プログラムのソースコードを示す図である。
【図４８】本実施例のブロック選別プログラムのソースコードを示す図である。
【図４９】本実施例のブロック選別プログラムのソースコードを示す図である。
【図５０】本実施例のブロック選別プログラムのソースコードを示す図である。
【図５１】本実施例のブロック選別プログラムのソースコードを示す図である。
【図５２】本実施例のブロック選別プログラムのソースコードを示す図である。
【図５３】本実施例のブロック選別プログラムのソースコードを示す図である。
【図５４】本実施例のブロック選別プログラムのソースコードを示す図である。
【図５５】本実施例のブロック選別プログラムのソースコードを示す図である。
【図５６】本実施例のブロック選別プログラムのソースコードを示す図である。
【図５７】本実施例のブロック選別プログラムのソースコードを示す図である。
【図５８】本実施例のブロック選別プログラムのソースコードを示す図である。
【図５９】本実施例のブロック選別プログラムのソースコードを示す図である。
【図６０】本実施例のブロック選別プログラムのソースコードを示す図である。
【図６１】本実施例のブロック選別プログラムのソースコードを示す図である。
【図６２】本実施例のブロック選別プログラムのソースコードを示す図である。
【図６３】本実施例のブロック選別プログラムのソースコードを示す図である。
【図６４】本実施例のブロック選別プログラムのソース

コードを示す図である。
【図６５】本実施例のブロック選別プログラムのソースコードを示す図である。
【図６６】本実施例のブロック選別プログラムのソースコードを示す図である。
【図６７】本実施例のブロック選別プログラムのソースコードを示す図である。
【図６８】本実施例のブロック選別プログラムのソースコードを示す図である。
【図６９】本実施例のブロック選別プログラムのソースコードを示す図である。
【図７０】本実施例のブロック選別プログラムのソースコードを示す図である。
【図７１】本実施例のブロック選別プログラムのソースコードを示す図である。
【図７２】本実施例のブロック選別プログラムのソースコードを示す図である。
【図７３】本実施例のブロック選別プログラムのソースコードを示す図である。
【図７４】本実施例のブロック選別プログラムのソースコードを示す図である。
【図７５】本実施例のブロック選別プログラムのソースコードを示す図である。
【図７６】本実施例のブロック選別プログラムのソースコードを示す図である。
【図７７】本実施例のブロック選別プログラムのソースコードを示す図である。
【図７８】本実施例のブロック選別プログラムのソースコードを示す図である。
【図７９】本実施例のブロック選別プログラムのソースコードを示す図である。
【図８０】本実施例のブロック選別プログラムのソースコードを示す図である。
【図８１】本実施例のブロック選別プログラムのソースコードを示す図である。
【図８２】本実施例のブロック選別プログラムのソースコードを示す図である。
【図８３】本実施例のブロック選別プログラムのソースコードを示す図である。
【図８４】本実施例のブロック選別プログラムのソースコードを示す図である。
【図８５】本実施例のブロック選別プログラムのソースコードを示す図である。
【図８６】本実施例のブロック選別プログラムのソースコードを示す図である。
【図８７】本実施例のブロック選別プログラムのソースコードを示す図である。
【図８８】本実施例のブロック選別プログラムのソースコードを示す図である。
【図８９】本実施例のブロック選別プログラムのソースコードを示す図である。
【図９０】本実施例のブロック選別プログラムのソースコードを示す図である。
【図９１】本実施例のブロック選別プログラムのソースコードを示す図である。
【図９２】本実施例のブロック選別プログラムのソースコードを示す図である。
【図９３】本実施例のブロック選別プログラムのソースコードを示す図である。
【図９４】本実施例のブロック選別プログラムのソースコードを示す図である。
【図９５】本実施例のブロック選別プログラムのソースコードを示す図である。
【図９６】本実施例のブロック選別プログラムのソースコードを示す図である。
【図９７】本実施例のブロック選別プログラムのソースコードを示す図である。
【図９８】本実施例のブロック選別プログラムのソースコードを示す図である。
【図９９】本実施例のブロック選別プログラムのソースコードを示す図である。
【図１００】本実施例のブロック選別プログラムのソースコードを示す図である。
【図１０１】本実施例のブロック選別プログラムのソースコードを示す図である。
【図１０２】本実施例のブロック選別プログラムのソースコードを示す図である。
【図１０３】本実施例のブロック選別プログラムのソースコードを示す図である。
【図１０４】本実施例のブロック選別プログラムのソースコードを示す図である。
【図１０５】本実施例のブロック選別プログラムのソースコードを示す図である。
【図１０６】本実施例のブロック選別プログラムのソースコードを示す図である。
【図１０７】本実施例のブロック選別プログラムのソースコードを示す図である。
【図１０８】本実施例のブロック選別プログラムのソースコードを示す図である。
【図１０９】本実施例のブロック選別プログラムのソースコードを示す図である。
【図１１０】本実施例のブロック選別プログラムのソースコードを示す図である。
【図１１１】本実施例のブロック選別プログラムのソースコードを示す図である。
【図１１２】本実施例のブロック選別プログラムのソースコードを示す図である。
【図１１３】本実施例のブロック選別プログラムのソースコードを示す図である。
【図１１４】本実施例のブロック選別プログラムのソー

スコードを示す図である。
【図１１５】本実施例のブロック選別プログラムのソースコードを示す図である。
【図１１６】本実施例のブロック選別プログラムのソースコードを示す図である。
【図１１７】本実施例のブロック選別プログラムのソースコードを示す図である。
【図１１８】本実施例のブロック選別プログラムのソースコードを示す図である。
【図１１９】本実施例のブロック選別プログラムのソースコードを示す図である。
【図１２０】本実施例のブロック選別プログラムのソースコードを示す図である。
【図１２１】本実施例のブロック選別プログラムのソースコードを示す図である。
【図１２２】本実施例のブロック選別プログラムのソースコードを示す図である。
【図１２３】本実施例のブロック選別プログラムのソースコードを示す図である。
【図１２４】本実施例のブロック選別プログラムのソースコードを示す図である。
【図１２５】本実施例のブロック選別プログラムのソースコードを示す図である。
【図１２６】本実施例のブロック選別プログラムのソースコードを示す図である。
【図１２７】本実施例のブロック選別プログラムのソースコードを示す図である。
【図１２８】本実施例のブロック選別プログラムのソースコードを示す図である。
【図１２９】本実施例のブロック選別プログラムのソースコードを示す図である。
【図１３０】本実施例のブロック選別プログラムのソースコードを示す図である。
【図１３１】本実施例のブロック選別プログラムのソースコードを示す図である。
【図１３２】本実施例のブロック選別プログラムのソースコードを示す図である。
【図１３３】本実施例のブロック選別プログラムのソースコードを示す図である。
【図１３４】本実施例のブロック選別プログラムのソースコードを示す図である。
【図１３５】本実施例のブロック選別プログラムのソースコードを示す図である。
【図１３６】本実施例のブロック選別プログラムのソースコードを示す図である。
【図１３７】本実施例の比例スペースプログラムのソースコードを示す図である。
【図１３８】本実施例の比例スペースプログラムのソースコードを示す図である。
【図１３９】本実施例の比例スペースプログラムのソースコードを示す図である。
【図１４０】本実施例の比例スペースプログラムのソースコードを示す図である。
【図１４１】本実施例の比例スペースプログラムのソースコードを示す図である。
【図１４２】本実施例の比例スペースプログラムのソースコードを示す図である。
【図１４３】本実施例の比例スペースプログラムのソースコードを示す図である。
【図１４４】本実施例の比例スペースプログラムのソースコードを示す図である。
【図１４５】本実施例の比例スペースプログラムのソースコードを示す図である。
【図１４６】本実施例の比例スペースプログラムのソースコードを示す図である。
【図１４７】本実施例の比例スペースプログラムのソースコードを示す図である。
【図１４８】本実施例の比例スペースプログラムのソースコードを示す図である。
【図１４９】本実施例の比例スペースプログラムのソースコードを示す図である。
【図１５０】本実施例の比例スペースプログラムのソースコードを示す図である。
【図１５１】本実施例の比例スペースプログラムのソースコードを示す図である。
【図１５２】本実施例の比例スペースプログラムのソースコードを示す図である。
【図１５３】本実施例の比例スペースプログラムのソースコードを示す図である。
【図１５４】本実施例の比例スペースプログラムのソースコードを示す図である。
【図１５５】本実施例の比例スペースプログラムのソースコードを示す図である。
【図１５６】本実施例の比例スペースプログラムのソースコードを示す図である。
【図１５７】本実施例の比例スペースプログラムのソースコードを示す図である。
【図１５８】本実施例の比例スペースプログラムのソースコードを示す図である。
【図１５９】本実施例の比例スペースプログラムのソースコードを示す図である。
【図１６０】本実施例の比例スペースプログラムのソースコードを示す図である。
【図１６１】本実施例の比例スペースプログラムのソースコードを示す図である。
【図１６２】本実施例の比例スペースプログラムのソースコードを示す図である。
【図１６３】本実施例の比例スペースプログラムのソースコードを示す図である。
【図１６４】本実施例の比例スペースプログラムのソー

スコードを示す図である。
【図１６５】本実施例の比例スペースプログラムのソースコードを示す図である。
【図１６６】本実施例の比例スペースプログラムのソースコードを示す図である。
【図１６７】本実施例の比例スペースプログラムのソースコードを示す図である。
【図１６８】本実施例の比例スペースプログラムのソースコードを示す図である。
【図１６９】本実施例の比例スペースプログラムのソースコードを示す図である。
【図１７０】本実施例の比例スペースプログラムのソースコードを示す図である。
【図１７１】本実施例の比例スペースプログラムのソースコードを示す図である。
【図１７２】本実施例の比例スペースプログラムのソースコードを示す図である。
【図１７３】本実施例の比例スペースプログラムのソースコードを示す図である。
【図１７４】本実施例の比例スペースプログラムのソースコードを示す図である。
【図１７５】本実施例の比例スペースプログラムのソースコードを示す図である。
【図１７６】本実施例の比例スペースプログラムのソースコードを示す図である。
【図１７７】本実施例の比例スペースプログラムのソースコードを示す図である。
【図１７８】本実施例の比例スペースプログラムのソースコードを示す図である。
【図１７９】本実施例の比例スペースプログラムのソースコードを示す図である。
【図１８０】本実施例の比例スペースプログラムのソースコードを示す図である。
【図１８１】本実施例の比例スペースプログラムのソースコードを示す図である。
【図１８２】本実施例の比例スペースプログラムのソースコードを示す図である。
【図１８３】本実施例の比例スペースプログラムのソースコードを示す図である。
【図１８４】本実施例の比例スペースプログラムのソースコードを示す図である。
【図１８５】本実施例の比例スペースプログラムのソースコードを示す図である。
【図１８６】本実施例の比例スペースプログラムのソースコードを示す図である。
【図１８７】本実施例の比例スペースプログラムのソースコードを示す図である。
【図１８８】本実施例の比例スペースプログラムのソースコードを示す図である。
【図１８９】本実施例の比例スペースプログラムのソースコードを示す図である。
【図１９０】本実施例の比例スペースプログラムのソースコードを示す図である。
【図１９１】本実施例の比例スペースプログラムのソースコードを示す図である。
【図１９２】本実施例の比例スペースプログラムのソースコードを示す図である。
【図１９３】本実施例の比例スペースプログラムのソースコードを示す図である。
【図１９４】本実施例の比例スペースプログラムのソースコードを示す図である。
【図１９５】本実施例の比例スペースプログラムのソースコードを示す図である。
【図１９６】本実施例の比例スペースプログラムのソースコードを示す図である。
【図１９７】本実施例の比例スペースプログラムのソースコードを示す図である。
【図１９８】本実施例の比例スペースプログラムのソースコードを示す図である。
【図１９９】本実施例の比例スペースプログラムのソースコードを示す図である。
【図２００】本実施例の比例スペースプログラムのソースコードを示す図である。
【図２０１】本実施例の比例スペースプログラムのソースコードを示す図である。
【図２０２】本実施例の比例スペースプログラムのソースコードを示す図である。
【図２０３】本実施例の比例スペースプログラムのソースコードを示す図である。
【図２０４】本実施例の比例スペースプログラムのソースコードを示す図である。
【図２０５】本実施例の比例スペースプログラムのソースコードを示す図である。
【図２０６】本実施例の比例スペースプログラムのソースコードを示す図である。
【図２０７】本実施例の比例スペースプログラムのソースコードを示す図である。
【図２０８】本実施例の比例スペースプログラムのソースコードを示す図である。
【図２０９】本実施例の比例スペースプログラムのソースコードを示す図である。
【図２１０】本実施例の比例スペースプログラムのソースコードを示す図である。
【図２１１】本実施例の比例スペースプログラムのソースコードを示す図である。
【図２１２】本実施例の比例スペースプログラムのソースコードを示す図である。
【図２１３】本実施例の比例スペースプログラムのソースコードを示す図である。
【図２１４】本実施例の比例スペースプログラムのソー

スコードを示す図である。

【図２１５】本実施例の比例スペースプログラムのソースコードを示す図である。

【図２１６】本実施例の比例スペースプログラムのソースコードを示す図である。

【図２１７】本実施例の単一スペース（輪郭）分割プログラムのソースコードを示す図である。

【図２１８】本実施例の単一スペース（輪郭）分割プログラムのソースコードを示す図である。

【図２１９】本実施例の単一スペース（輪郭）分割プログラムのソースコードを示す図である。

【図２２０】本実施例の単一スペース（輪郭）分割プログラムのソースコードを示す図である。

【図２２１】本実施例の単一スペース（輪郭）分割プログラムのソースコードを示す図である。

【図２２２】本実施例の単一スペース（輪郭）分割プログラムのソースコードを示す図である。

【図２２３】本実施例の単一スペース（輪郭）分割プログラムのソースコードを示す図である。

【図２２４】本実施例の単一スペース（輪郭）分割プログラムのソースコードを示す図である。

【図２２５】本実施例の単一スペース（輪郭）分割プログラムのソースコードを示す図である。

【図２２６】本実施例の単一スペース（輪郭）分割プログラムのソースコードを示す図である。

【図２２７】本実施例の単一スペース（輪郭）分割プログラムのソースコードを示す図である。

【図２２８】本実施例の単一スペース（輪郭）分割プログラムのソースコードを示す図である。

【図２２９】本実施例の単一スペース（輪郭）分割プログラムのソースコードを示す図である。

【図２３０】本実施例の単一スペース（輪郭）分割プログラムのソースコードを示す図である。

【図２３１】本実施例の単一スペース（輪郭）分割プログラムのソースコードを示す図である。

【図２３２】本実施例の単一スペース（輪郭）分割プログラムのソースコードを示す図である。

【図２３３】本実施例の単一スペース（輪郭）分割プログラムのソースコードを示す図である。

【図２３４】本実施例の単一スペース（輪郭）分割プログラムのソースコードを示す図である。

【図２３５】本実施例の単一スペース（輪郭）分割プログラムのソースコードを示す図である。

【図２３６】本実施例の単一スペース（輪郭）分割プログラムのソースコードを示す図である。

【図２３７】本実施例の単一スペース（輪郭）分割プログラムのソースコードを示す図である。

【図２３８】本実施例の単一スペース（輪郭）分割プログラムのソースコードを示す図である。

【図２３９】本実施例の単一スペース（輪郭）分割プログラムのソースコードを示す図である。

【図２４０】本実施例の単一スペース（輪郭）分割プログラムのソースコードを示す図である。

【図２４１】本実施例の単一スペース（輪郭）分割プログラムのソースコードを示す図である。

【図２４２】本実施例の単一スペース（輪郭）分割プログラムのソースコードを示す図である。

【図２４３】本実施例の単一スペース（輪郭）分割プログラムのソースコードを示す図である。

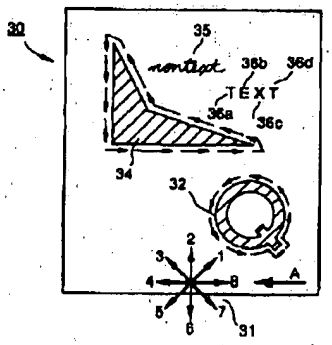
【図２４４】本実施例の単一スペース（輪郭）分割プログラムのソースコードを示す図である。

【図２４５】本実施例の単一スペース（輪郭）分割プログラムのソースコードを示す図である。

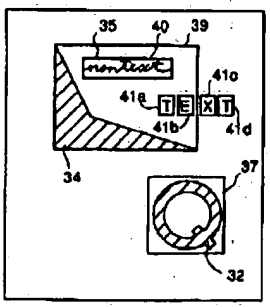
【図２４６】本実施例の単一スペース（輪郭）分割プログラムのソースコードを示す図である。

【図２４７】本実施例の単一スペース（輪郭）分割プログラムのソースコードを示す図である。

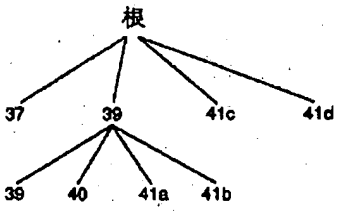
【図５Ａ】



【図５Ｂ】



【図５Ｃ】



【図２６Ｂ】



【図２６Ｃ】



【図１２６】

```
#define FILE_LENGTH          80
#define LINE_LENGTH          256
#define POINT_SOURCE_LENGTH  30
```

【図1】

【図36】


```
#include "UxLib.h"
main(argc,argv)
    unsigned int argc;
    char **argv;
{
    swidget sw, popup_blocksel();

    /*.....*/
    /*    Initialize Architect    */
    /*.....*/

    UxTopLevel = UxInitialize("OCR Block Selection Demo", &argc, argv);

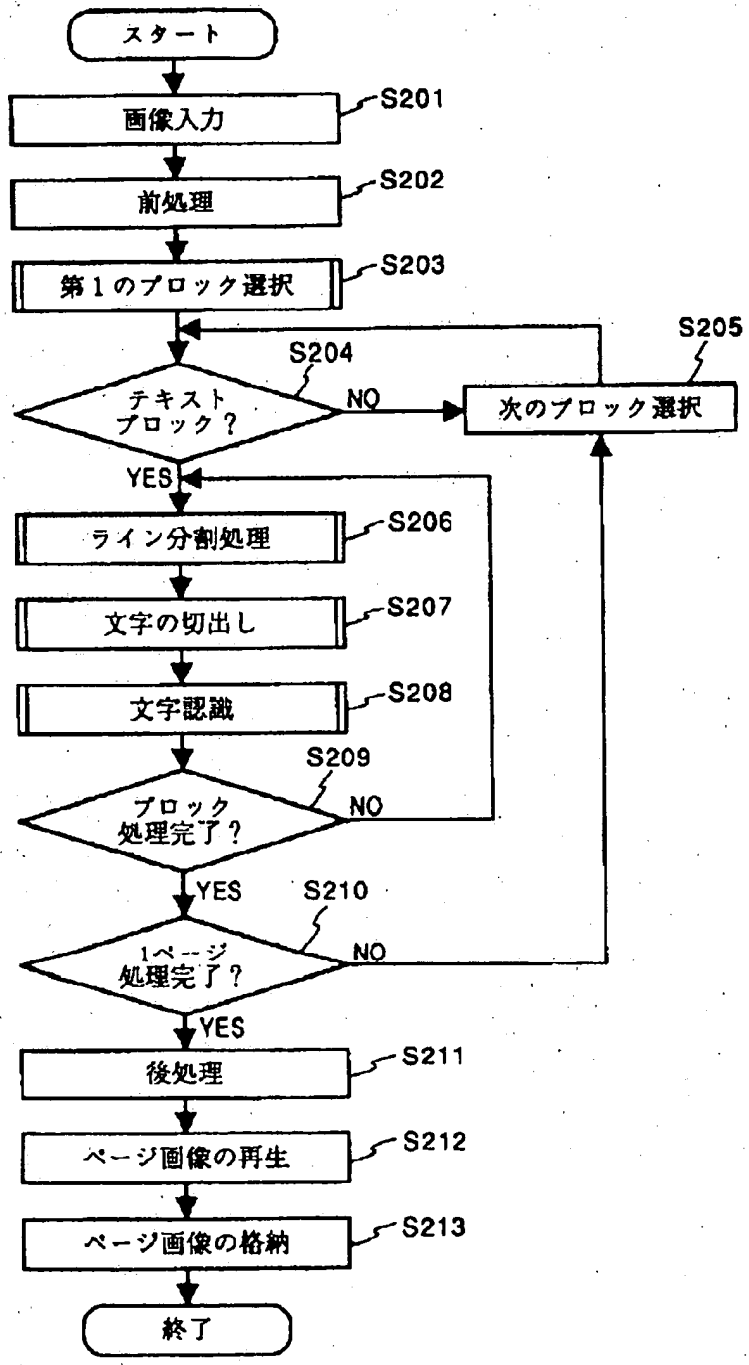
    /*.....*/
    /* Create and popup the first window of the interface.    */
    /*.....*/

    sv = popup_blocksel();

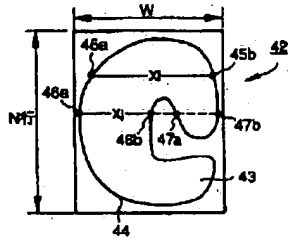
    /*.....*/
    /*    Enter the event loop; this function never returns.    */
    /*.....*/

    UxMainLoop();
}
```

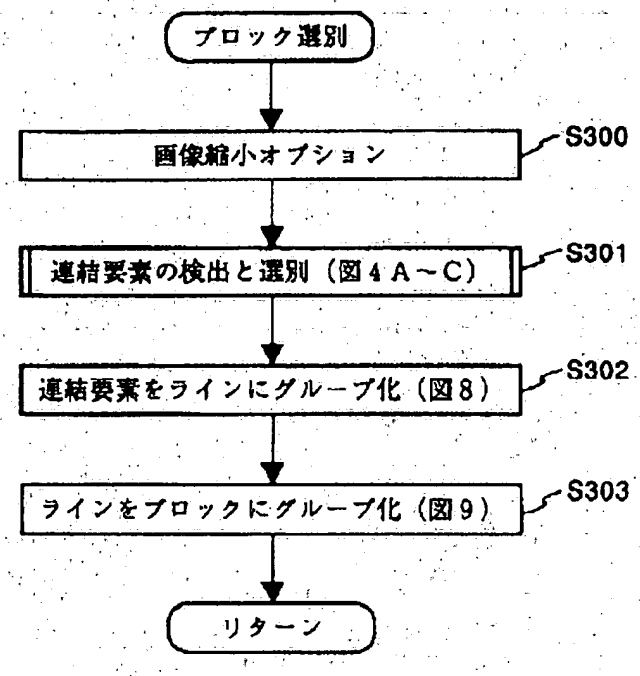
【図2】


スタート
画像入力
S201
前処理
S202
第1のブロック選択
S203
S204
テキスト
ブロック？
NO
S205
次のブロック選択
YES
ライン分割処理
S206
文字の切出し
S207
文字認識
S208
S209
ブロック
処理完了？
NO
YES
S210
1ページ
処理完了？
NO
YES
後処理
S211
ページ画像の再生
S212
ページ画像の格納
S213
終了

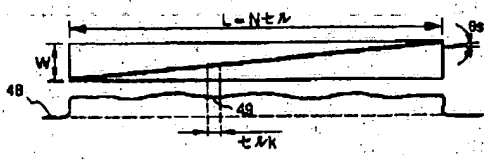
【図6A】


W
45a
45b
42
Xt
46a
Nlf
Xj
47b
46b
47a
43
44

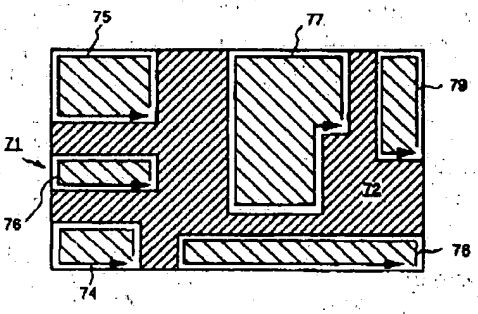
【図3】



【図6B】



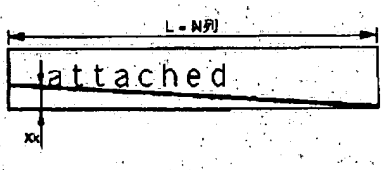
【図7B】



【図194】

```
free((char **) N);
}
```

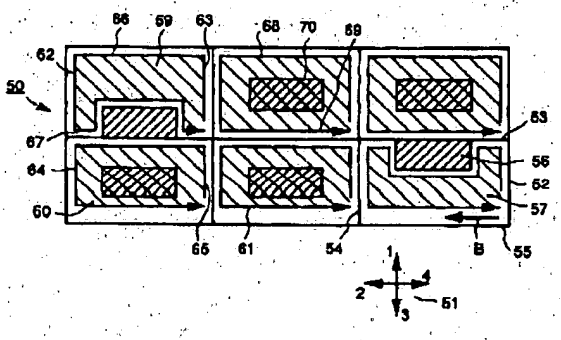
【図6C】



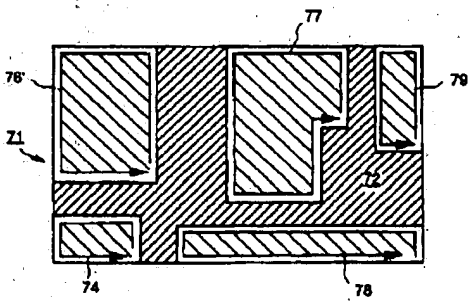
【図234】

```
free((char **) N);
}
```

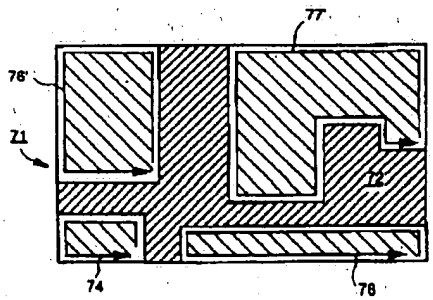
【図7A】



【図7C】



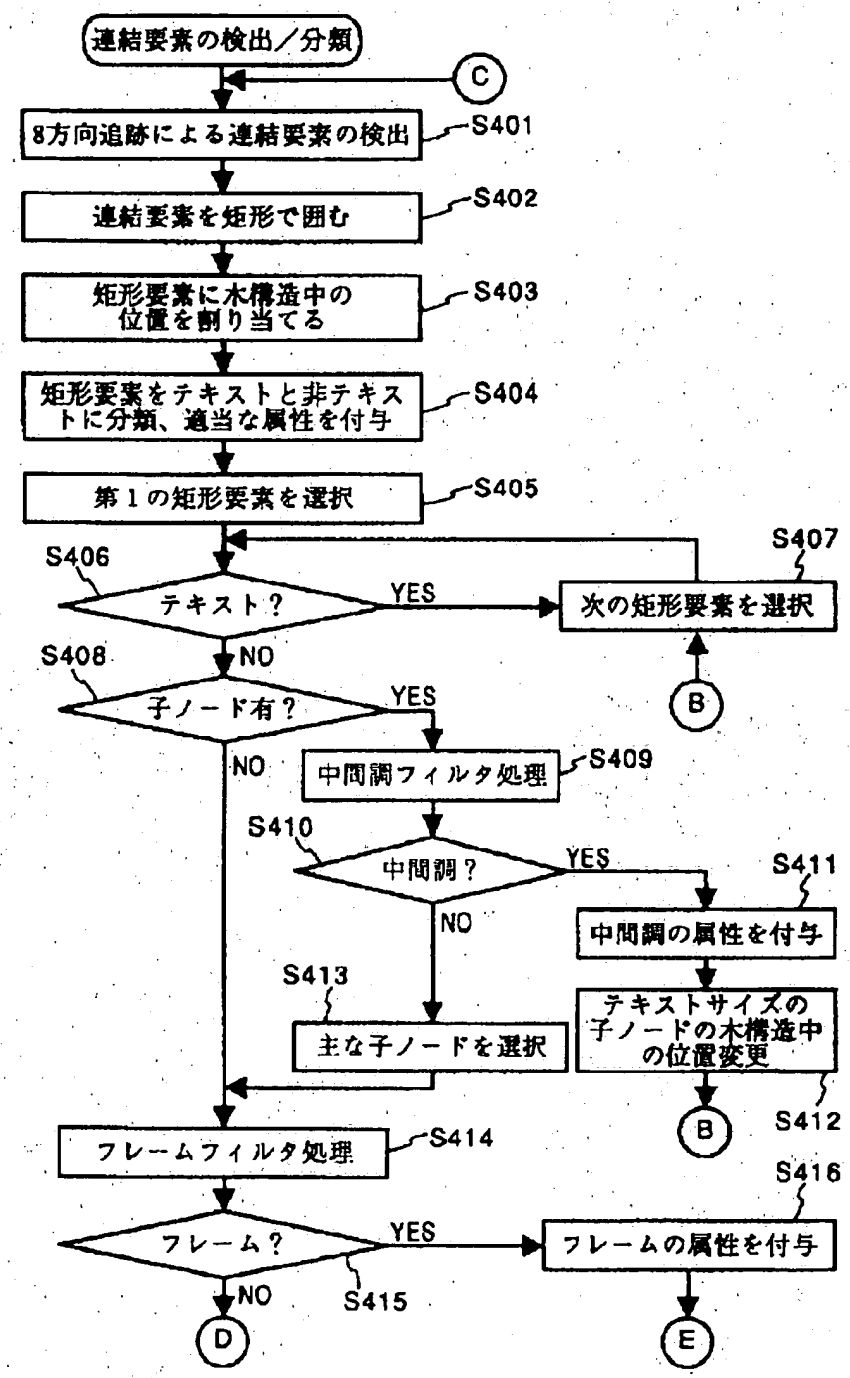
【図7D】



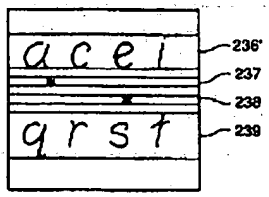
【図26D】

f y

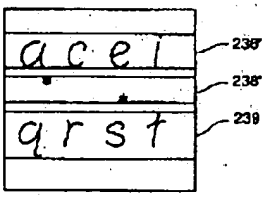
【図４Ａ】



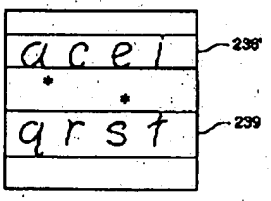
【図１８Ｂ】



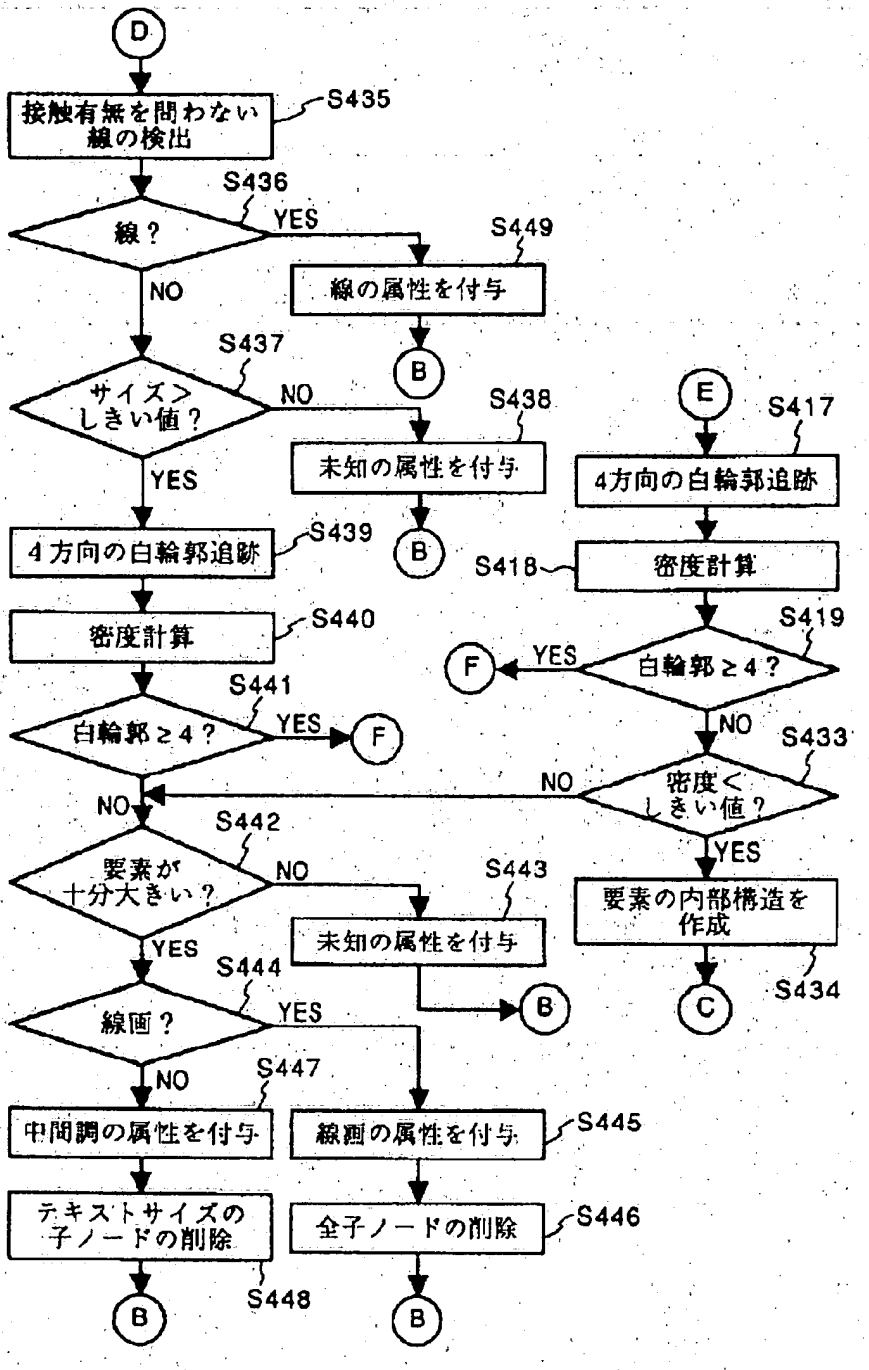
【図１８Ｃ】



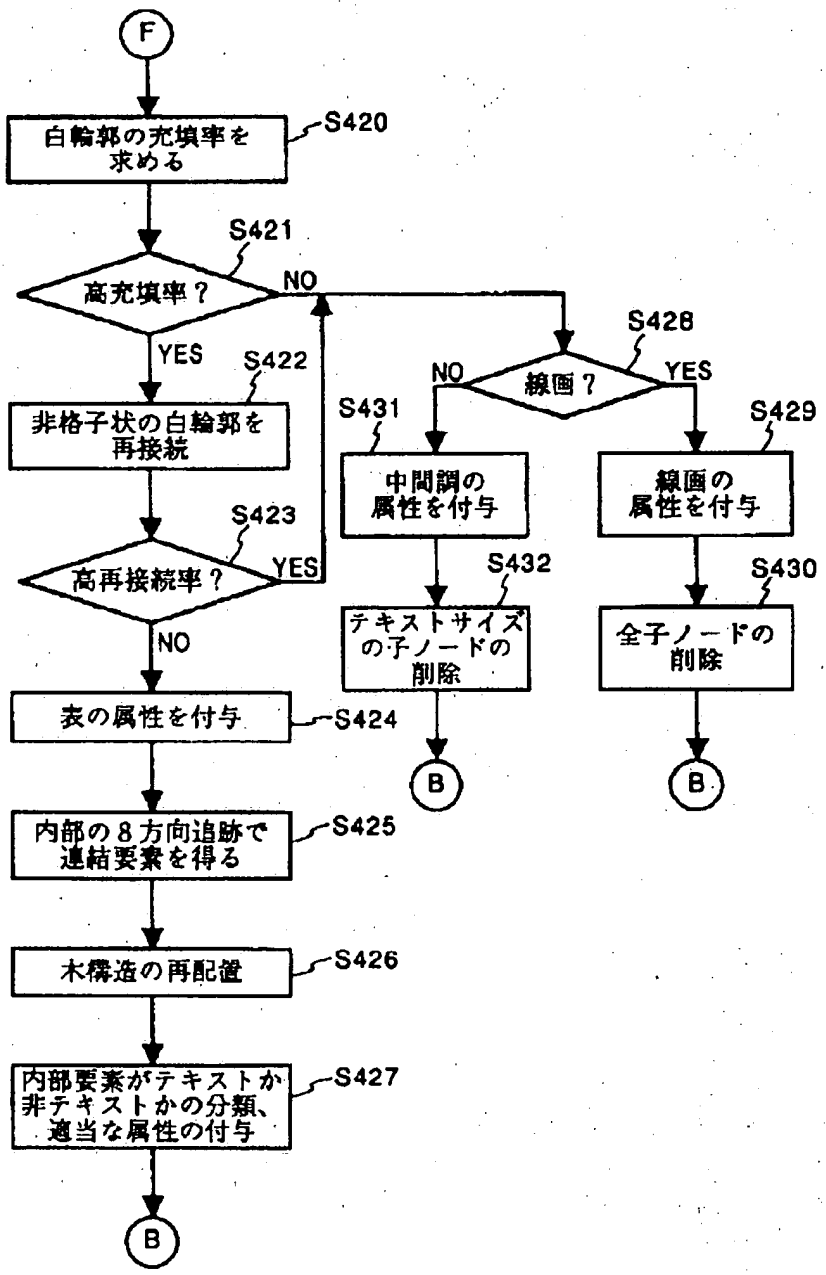
【図１８Ｄ】



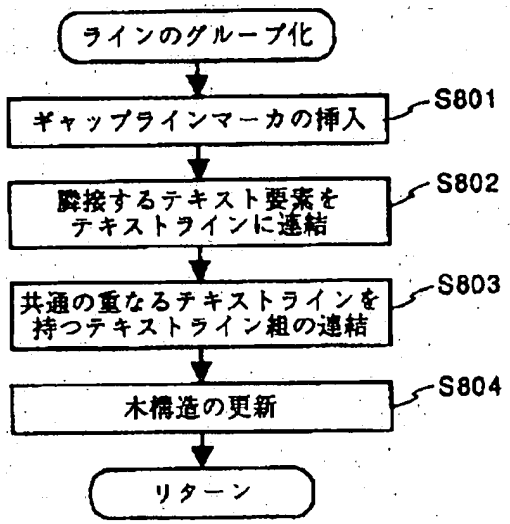
【図4B】


D
接触有無を問わない線の検出
S435
S436
線？
YES
S449
線の属性を付与
NO
S437
サイズ＞しきい値？
NO
S438
未知の属性を付与
YES
4方向の白輪郭追跡
S439
密度計算
S440
白輪郭≧4？
S441
YES
F
NO
要素が十分大きい？
S442
NO
S443
未知の属性を付与
YES
線画？
S444
YES
NO
S447
中間調の属性を付与
テキストサイズの子ノードの削除
S448
線画の属性を付与
S445
全子ノードの削除
S446
B
E
S417
4方向の白輪郭追跡
S418
密度計算
S419
白輪郭≧4？
YES
F
NO
S433
密度＜しきい値？
NO
YES
要素の内部構造を作成
S434
C

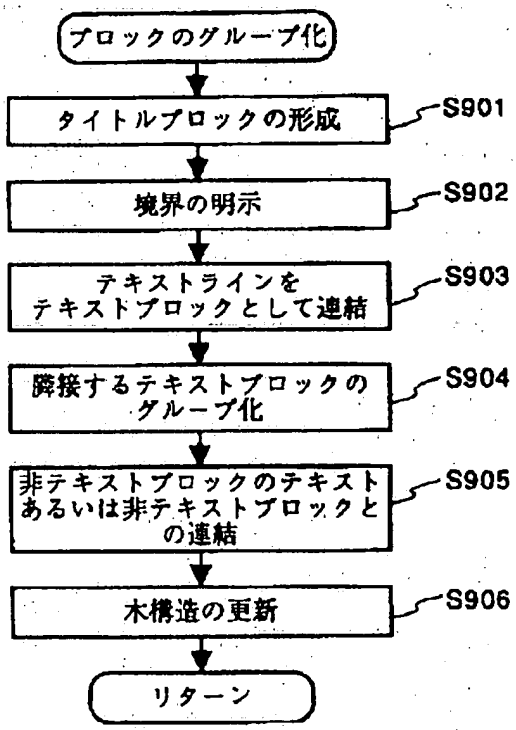
【図4C】


F
白輪郭の充填率を求める
S420
S421
高充填率？
NO
YES
S422
非格子状の白輪郭を再接続
S423
高再接続率？
YES
NO
表の属性を付与
S424
内部の8方向追跡で連結要素を得る
S425
木構造の再配置
S426
内部要素がテキストか非テキストかの分類、適当な属性の付与
S427
B
S428
線画？
NO
YES
S431
中間調の属性を付与
S432
テキストサイズの子ノードの削除
B
S429
線画の属性を付与
S430
全子ノードの削除
B

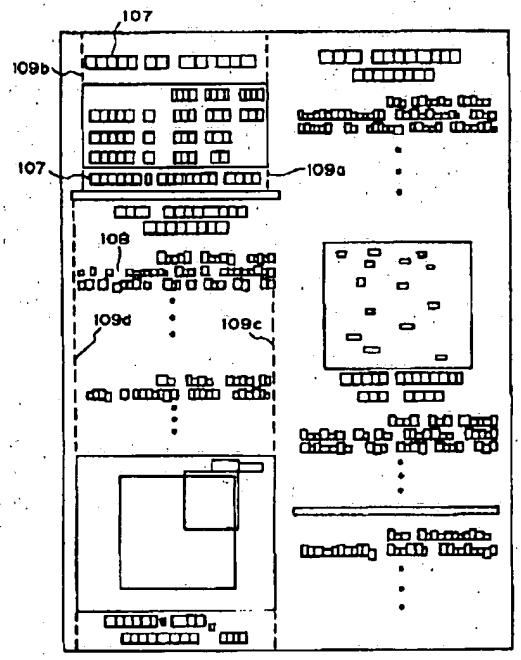
【図8】


ラインのグループ化
ギャップラインマーカの挿入
S801
隣接するテキスト要素を
テキストラインに連結
S802
共通の重なるテキストラインを
持つテキストライン組の連結
S803
木構造の更新
S804
リターン

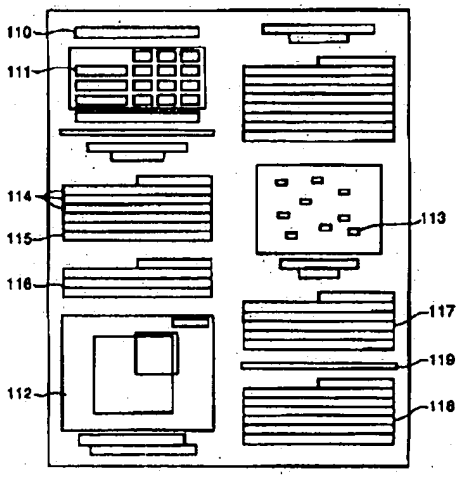
【図9】


ブロックのグループ化
タイトルブロックの形成
S901
境界の明示
S902
テキストラインを
テキストブロックとして連結
S903
隣接するテキストブロックの
グループ化
S904
非テキストブロックのテキスト
あるいは非テキストブロックと
の連結
S905
木構造の更新
S906
リターン

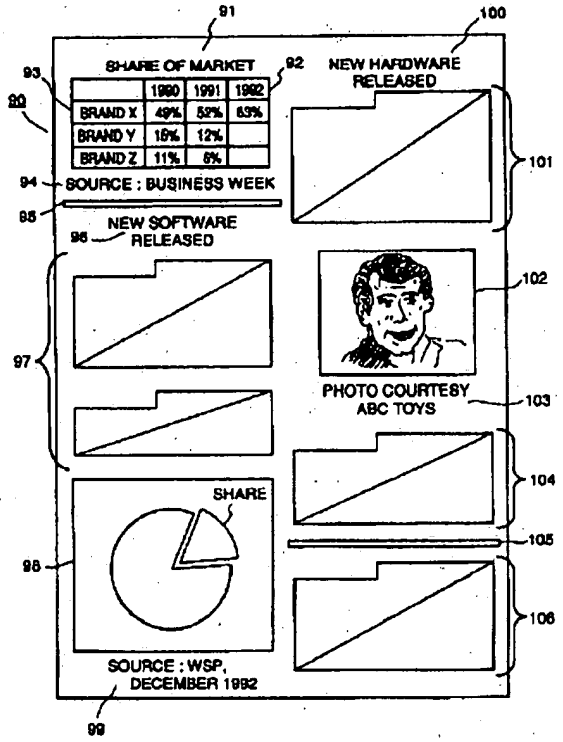
【図11】


107
109b
107
109a
108
109d
109c

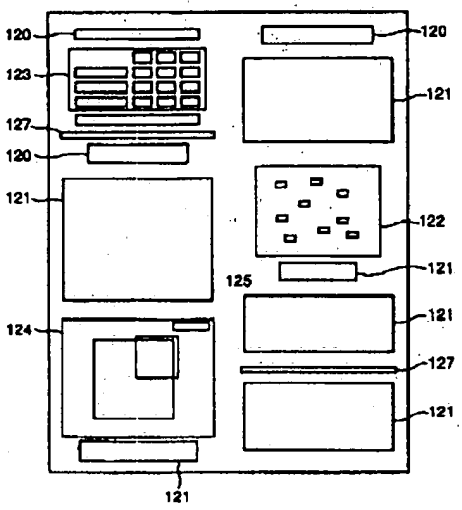
【図12】


110
111
114
115
116
112
113
117
119
118

【図１０】



【図１３】



【図２６Ａ】

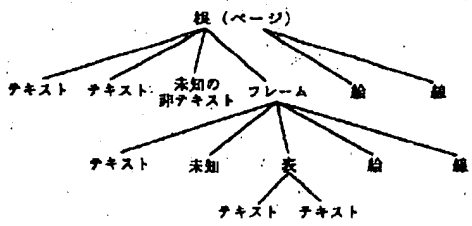


【図３５】

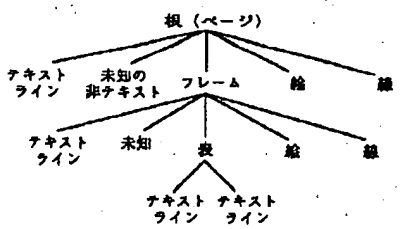
Source Code For

Block Selection

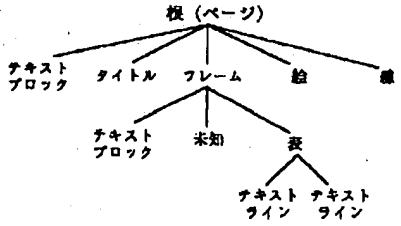
【図１４】



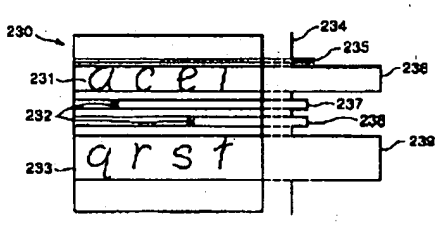
【図１５】



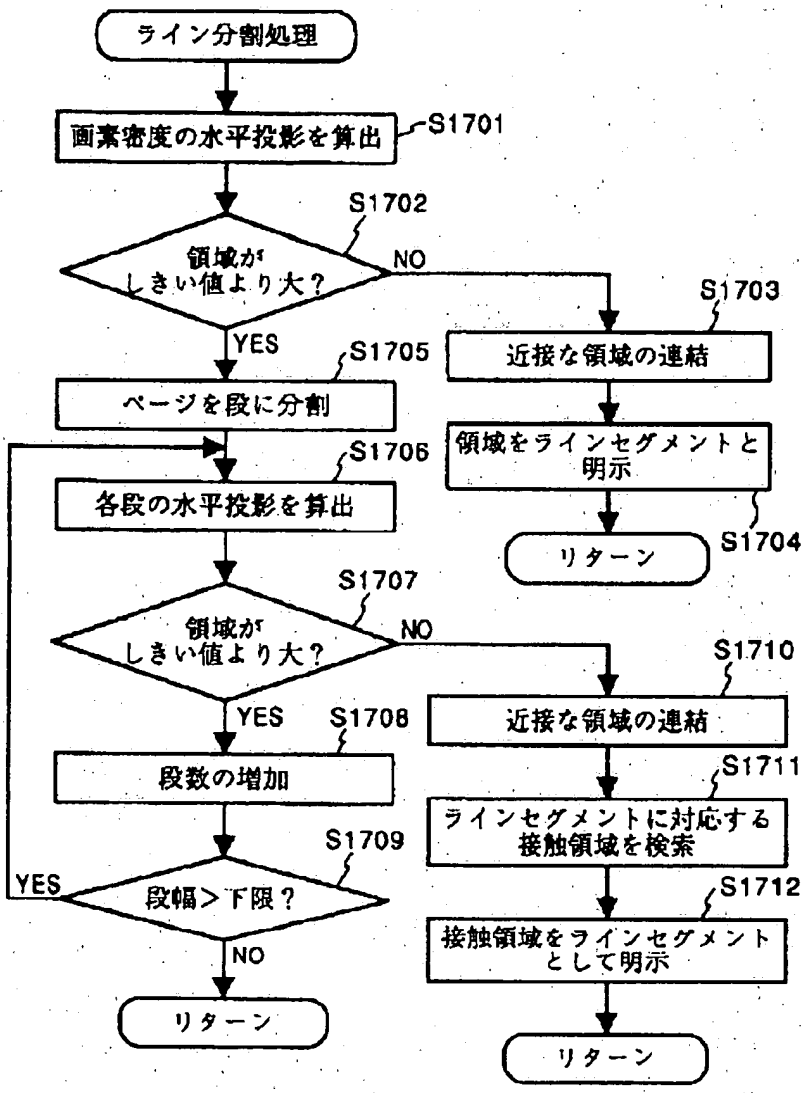
【図１６】



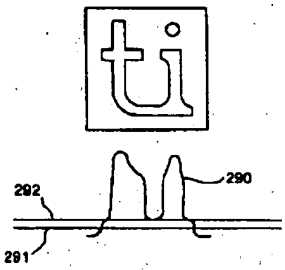
【図１８Ａ】



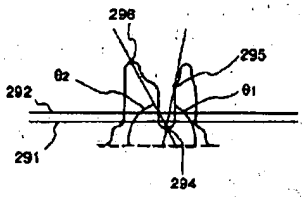
【図17】



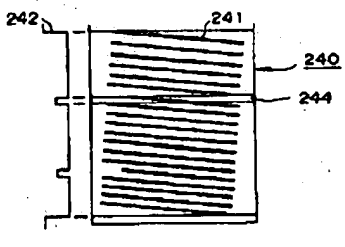
【図29A】



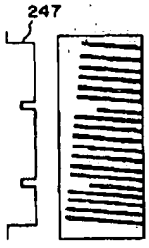
【図29B】



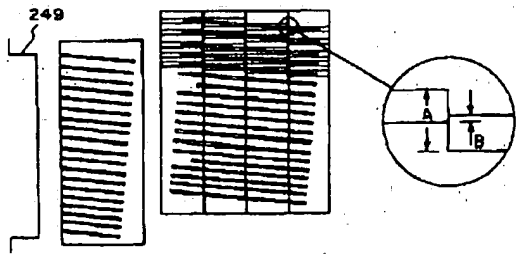
【図19A】



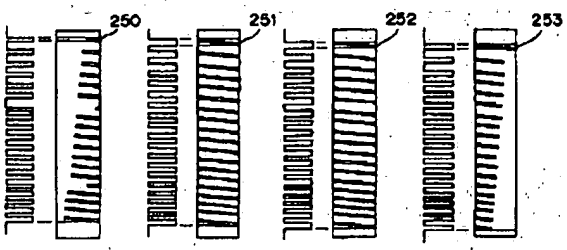
【図19B】



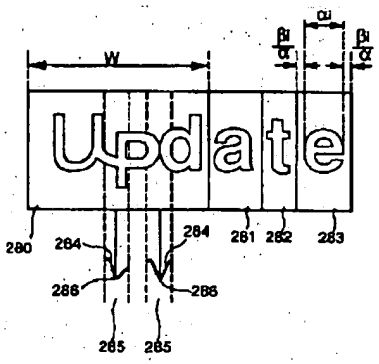
【図19D】



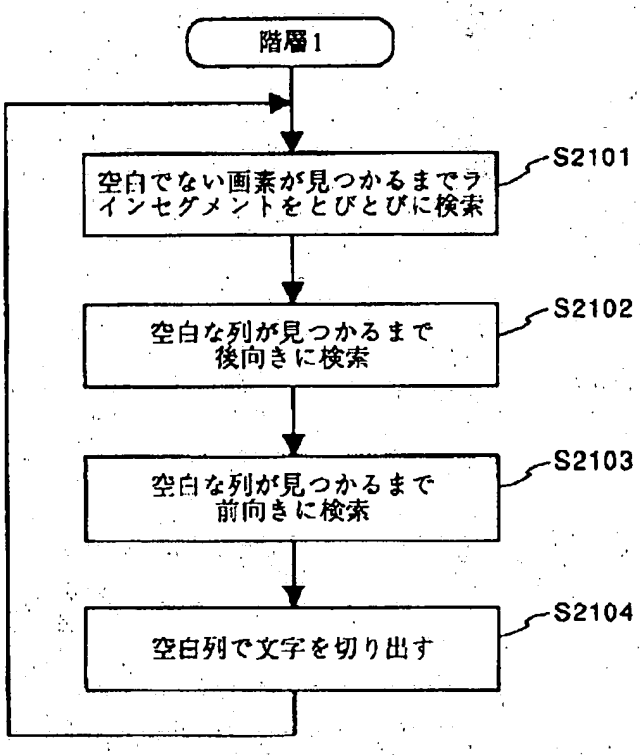
【図１９Ｃ】



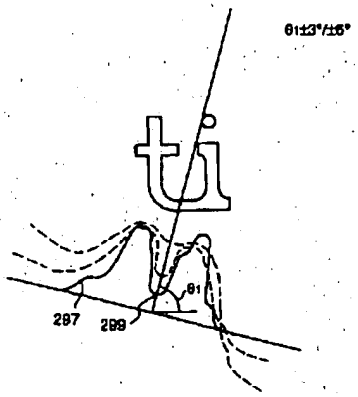
【図２４】



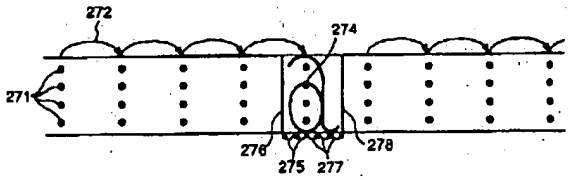
【図２１】



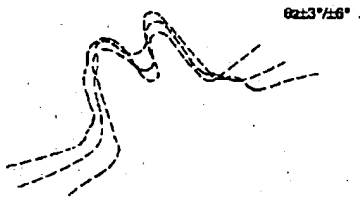
【図２９Ｃ】



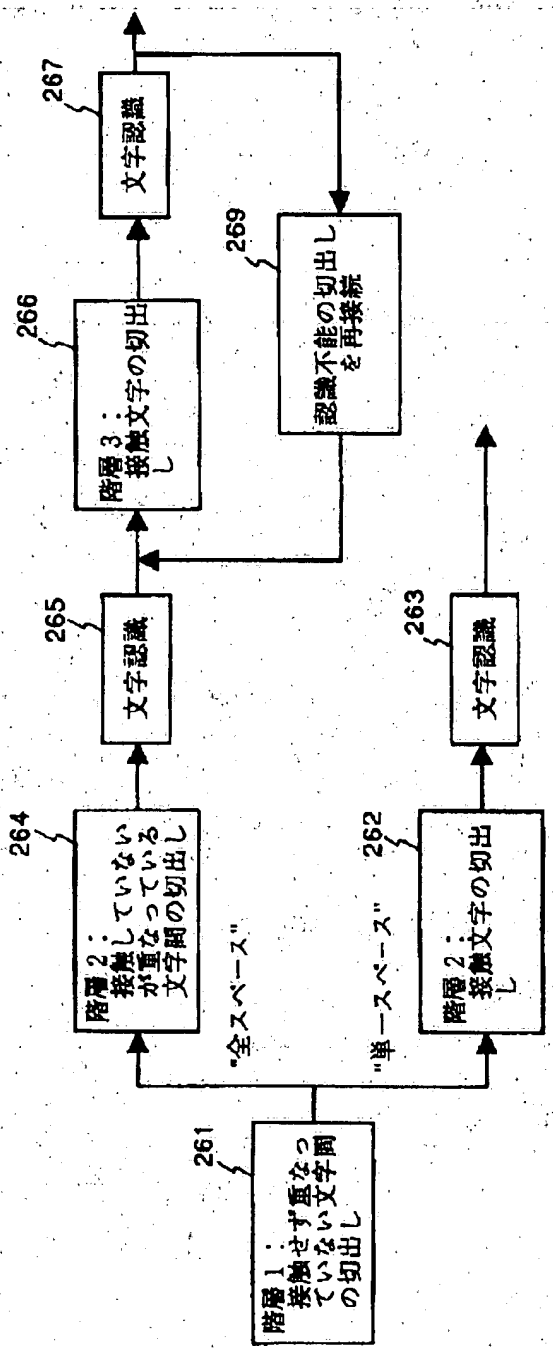
【図２２】



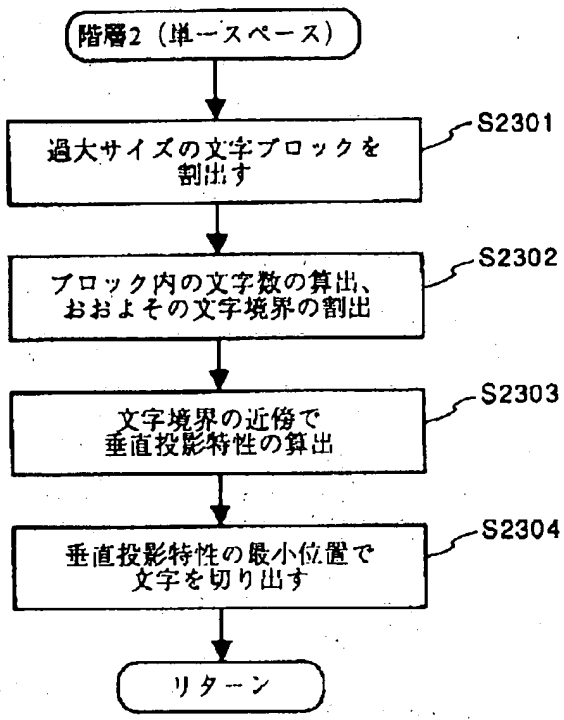
【図２９Ｄ】



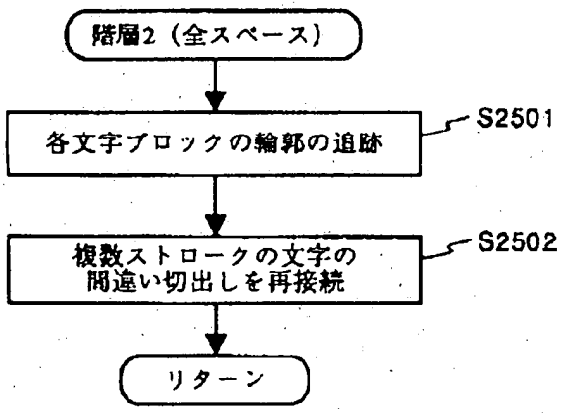
【図２０】


261
階層１：接触せず重なっていない文字間の切出し
"全スペース"
"単一スペース"
264
階層２：接触していないが重なっている文字間の切出し
262
階層２：接触文字の切出し
265
文字認識
263
文字認識
266
階層３：接触文字の切出し
269
認識不能の切出しを再接続
267
文字認識

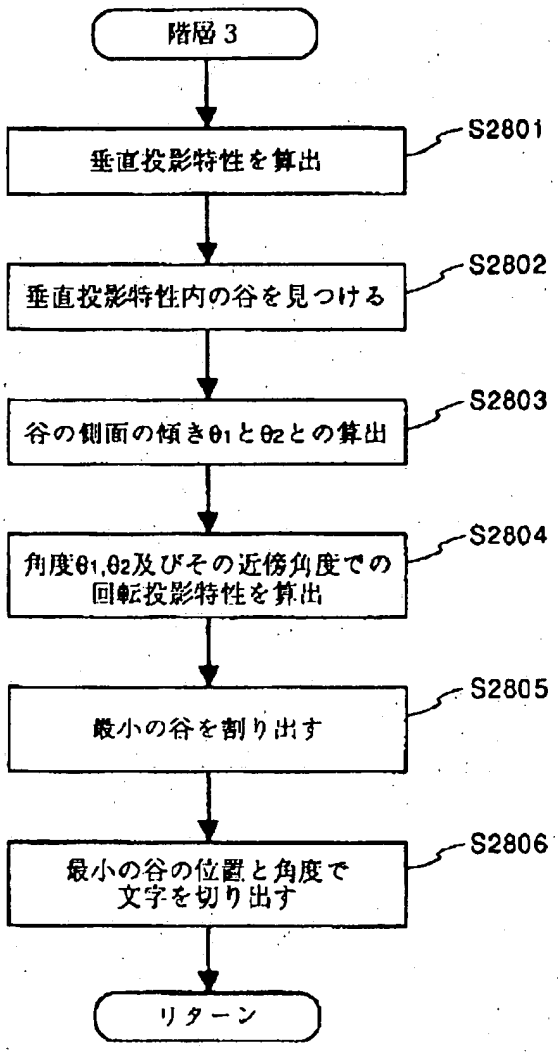
【図２３】



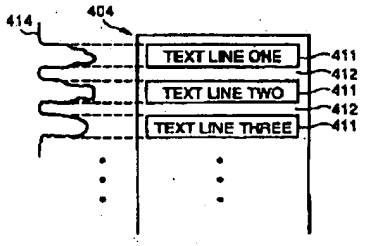
【図２５】



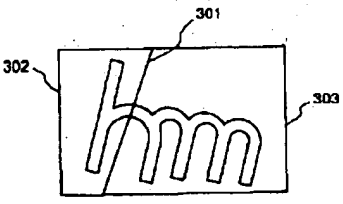
【図２８】



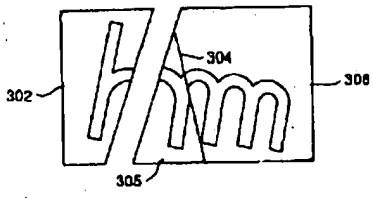
【図３３Ａ】



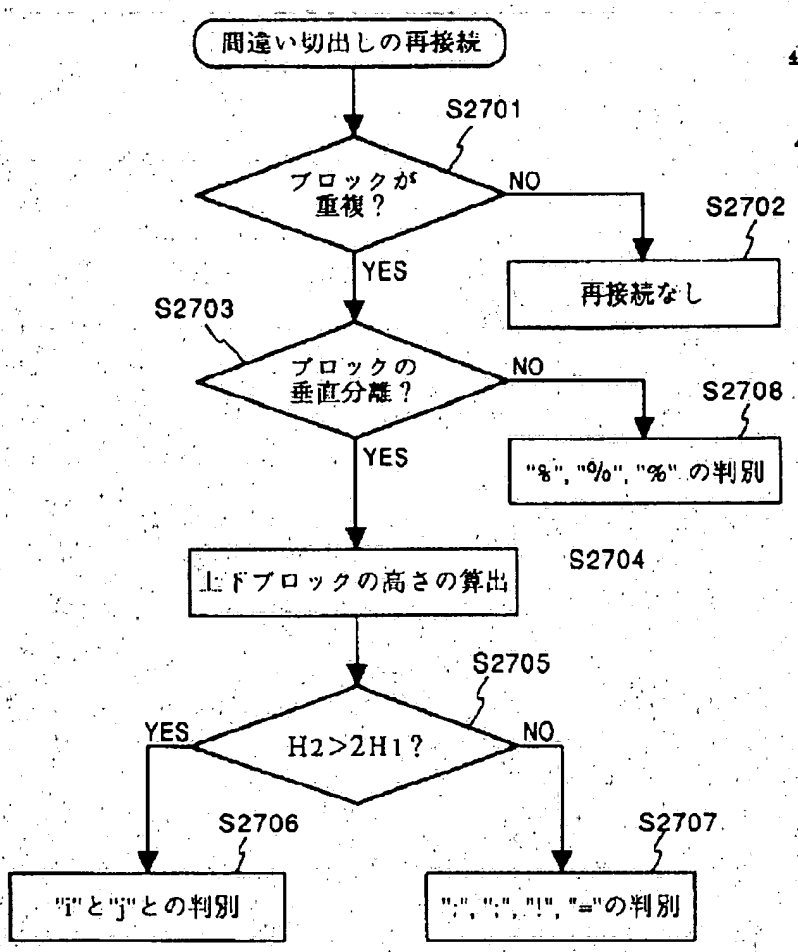
【図３１Ａ】



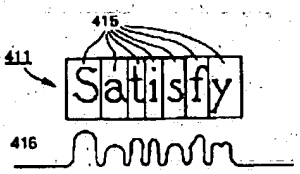
【図３１Ｂ】



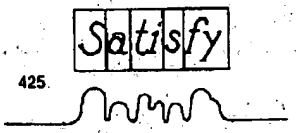
【図27】



【図34A】



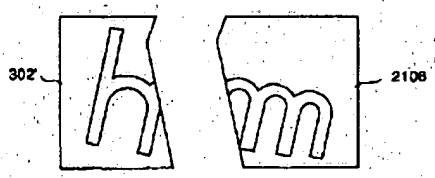
【図34B】



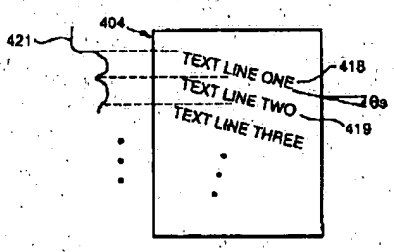
【図134】

```
extern   float *vector();
extern   unsigned char *cvector();
extern   int *ivector();
extern   double *dvector();
extern   unsigned char **cmatrix();
extern   float **matrix();
extern   int **imatrix();
extern   double **dmatrix();
extern   void free_vector();
extern   void free_ivector();
extern   void free_dvector();
extern   void free_matrix();
extern   void free_imatrix();
extern   void free_dmatrix();
extern   void free_cmatrix();
```

【図31C】



【図33B】



【図129】

```
struct sectionblock *previous;
struct sectionblock *next;

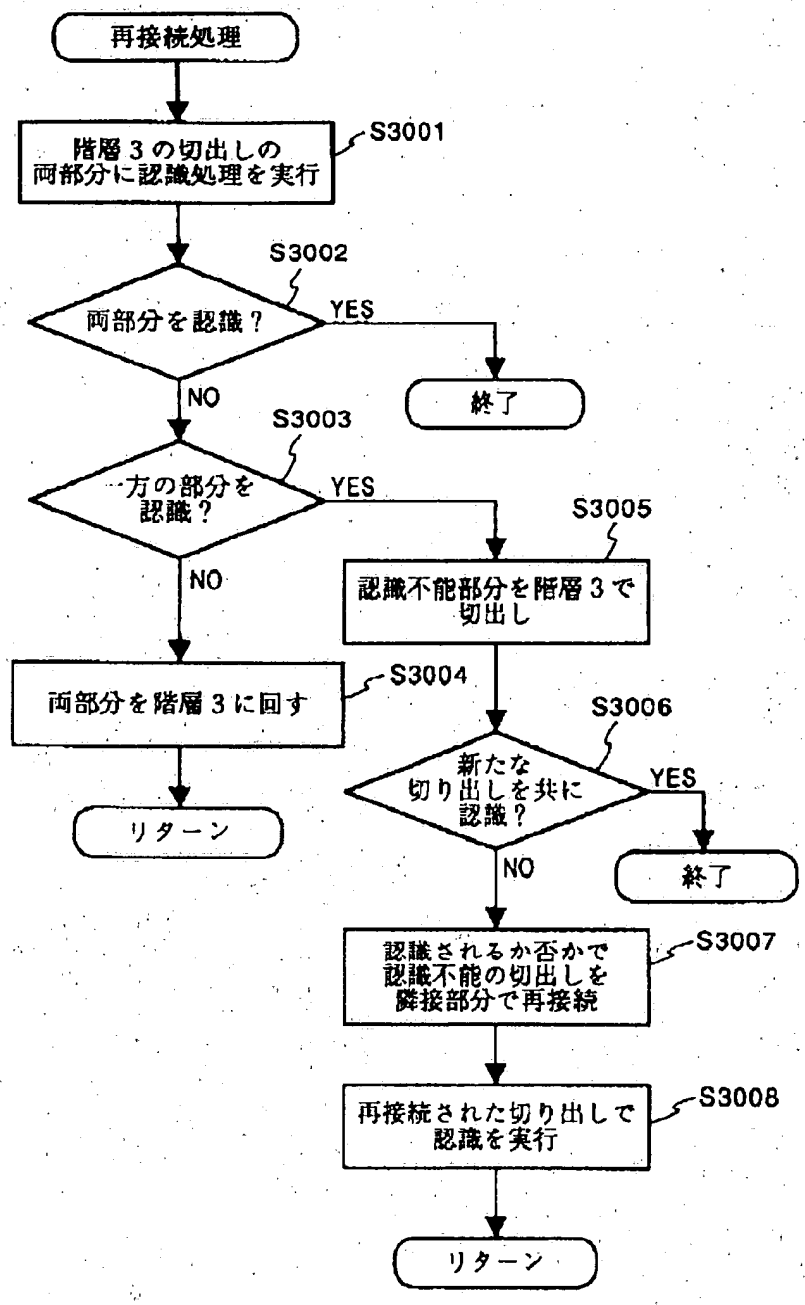
int   upper_y, lower_y, left_x, right_x;
int   width, length;
int   index;
int   numlineblock;
int   ste;
int   boundary_index;
};
```

【図163】

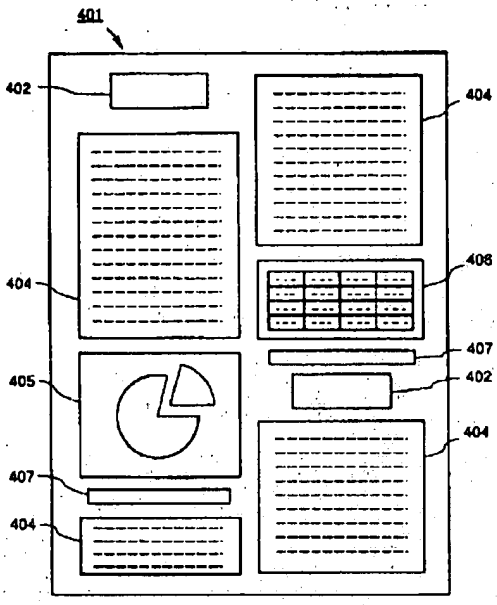
```
                    HISTORESIS_THRESHOLD);
  charcn = charcn->next;      // Get the next character
}

free_ivector(histogram);      // Deallocate the memory
}
```

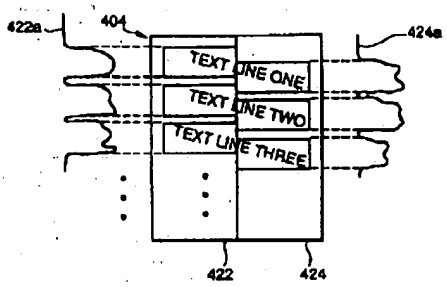
【図３０】


再接続処理
階層３の切出しの両部分に認識処理を実行
S3001
S3002
両部分を認識？
YES
終了
NO
S3003
一方の部分を認識？
YES
S3005
認識不能部分を階層３で切出し
NO
両部分を階層３に回す
S3004
S3006
新たな切り出しを共に認識？
YES
終了
リターン
NO
認識されるか否かで認識不能の切出しを隣接部分で再接続
S3007
再接続された切り出しで認識を実行
S3008
リターン

【図３２】



【図３３C】



【図６２】

```
int width, height;
{
    int x, y, xdis, ydis, i;
    static fid = 13;

    init_graphics(wd);
    xs_clr_win(wd);
    xdis = ((height-10)/(TOTALCOLOR));
    xs_getfont(wd);
    for ( i = 0; i < TOTALCOLOR; i++){
        xs_set_foreground(color_code[i], wd);
        xs_setfont(wd, fid);
        xs_string(wd, 20, (i+1)*xdis+20, color_label[i]);
    }
    xs_flush(wd);
}
```

【図４４】

```
/* ARGSUSED */
static void okCallback_QuitMessageBox(UxWidget, UxContext, UxCallbackArg)
        Widget          UxWidget;
        _UxCQuitDialog  *UxContext;
        int             UxCallbackArg;
{
        swidget         UxThisWidget;

        UxThisWidget = UxWidgetToSwidget(UxWidget);
        _UxFromContext(UxContext);

        {
          exit();
        }

        _UxToContext(UxContext);
}
```

【図１３３】

```
#define R               0
#define G               1
#define B               2

int color_comp[9][3]={{0x43, 0xcb, 0x00},
                      {0xfe, 0x00, 0x00},
                      {0xfe, 0x00, 0xfe},
                      {0xfe, 0xfe, 0x00},
                      {0xfe, 0x98, 0x00},
                      {0x9e, 0x00, 0x00},
                      {0x45, 0xfe, 0xfe},
                      {0x45, 0x00, 0xfe},
                      {0x4b, 0xee, 0x98}};

/*
int color_comp[9][3]={{0x00, 0x00, 0x00},
                      {0x00, 0x00, 0x00},
                      {0x00, 0x00, 0x00},
                      {0x00, 0x00, 0x00},
                      {0x00, 0x00, 0x00},
                      {0x00, 0x00, 0x00},
                      {0x00, 0x00, 0x00},
                      {0x00, 0x00, 0x00},
                      {0x00, 0x00, 0x00}};

*/
```

【図４６】

```
}
        _UxCon->DisplayForm= DisplayForm;
        _UxCon->DisplayFormClose_PBG= DisplayFormClose_PBG;
        _UxCon->DisplayForm_DrawArea= DisplayForm_DrawArea;
}

static void _UxFromContext(_UxCon)
        _UxCDisplayForm *_UxCon;
{
        DisplayForm= _UxCon->DisplayForm;
        DisplayFormClose_PBG= _UxCon->DisplayFormClose_PBG;
        DisplayForm_DrawArea= _UxCon->DisplayForm_DrawArea;
}

/* ARGSUSED */
static void activateCallback_DisplayFormClose_PBG(UxWidget, UxContext, UxCallbackArg)
        Widget          UxWidget;
        _UxCDisplayForm *UxContext;
        int             UxCallbackArg;
{
        swidget         UxThisWidget;

        UxThisWidget = UxWidgetToSwidget(UxWidget);
        _UxFromContext(UxContext);

        {
            UxPopdownInterface (DisplayForm);
        }

        _UxToContext(UxContext);
}
```

【図37】

【図38】

```
scanButton= UxCreatePushButtonGadget('scanButton', fileP
fileFrame_b1= UxCreateRowColumn('fileFrame_b1', fileP
openButton= UxCreatePushButtonGadget('openButton', f
fileFrame_b2= UxCreateRowColumn('fileFrame_b2', fileP
viewButton= UxCreatePushButtonGadget('viewButton', f
fileFrame_b3= UxCreateRowColumn('fileFrame_b3', fileP
exitButton= UxCreatePushButtonGadget('exitButton', f
runFrame= UxCreateRowColumn('runFrame', pulldownMenu
runButton= UxCreatePushButtonGadget('runButton', runF
helpFrame= UxCreateRowColumn('helpFrame', pulldownMen
colorRef= UxCreatePushButtonGadget('colorRef', helpFra
fileCascade= UxCreateCascadeButton('fileCascade', pull
runCascade= UxCreateCascadeButton('runCascade', pullD
helpCascade= UxCreateCascadeButton('helpCascade', pull

_UxCpContext( _Uxcon);

init_BlockSel(BlockSel);

UxCreateWidget(BlockSel);

UxCreateWidget(mainWindow);

UxCreateWidget(BlockSelBB);

UxCreateWidget(README);

UxCreateWidget(pulldownMenu);

UxCreateWidget(filePane);

UxCreateWidget(scanButton);
UxAddCallback(scanButton, "activateCallback", activateCallback_scanButton, _U
    xCon);

UxCreateWidget(filePane_b1);

UxCreateWidget(openButton);
UxAddCallback(openButton, "activateCallback", activateCallback_openButton, _U
    xCon);

UxCreateWidget(filePane_b2);

UxCreateWidget(viewButton);
UxAddCallback(viewButton, "activateCallback", activateCallback_viewButton, _U
    xCon);

UxCreateWidget(filePane_b3);

UxCreateWidget(exitButton);
UxAddCallback(exitButton, "activateCallback", activateCallback_exitButton, _U
    xCon);

UxCreateWidget(runMenu);

UxCreateWidget(runButton);
UxAddCallback(runButton, "activateCallback", activateCallback_runButton, _UxC
    on);

UxCreateWidget(helpPane);

UxCreateWidget(colorRef);
UxAddCallback(colorRef, "activateCallback", activateCallback_colorRef, _U
    xCon);

UxPutSubMenuId(fileCascade, "filePane");
UxCreateWidget(fileCascade);

UxPutRowColumnType(runPane, "menu_pulldown");
UxCreateWidget(runCascade);

UxPutSubMenuId(helpCascade, "helpPane");
UxCreateWidget(helpCascade);

UxPutRowColumnType(helpPane, "menu_pulldown");
UxCreateWidget(colorRef);

UxPutLabelString(colorRef, "Color References...");
UxPutMnemonic(colorRef, "C");

UxPutLabelString(helpCascade, "Help");
UxPutMnemonic(helpCascade, "H");

UxPutSubMenuId(runCascade, "runPane");
UxCreateWidget(runButton);

UxPutLabelString(runButton, "Run");
UxPutMnemonic(runButton, "R");

UxPutLabelString(exitButton, "Exit");
UxPutMnemonic(exitButton, "E");

UxPutRowColumnType(runPane, "menu_pulldown");
UxCreateWidget(viewButton);

UxPutLabelString(viewButton, "View ...");
UxPutMnemonic(viewButton, "V");

UxPutLabelString(openButton, "Open ...");
UxPutMnemonic(openButton, "O");

UxPutLabelString(scanButton, "Scan ...");
UxPutMnemonic(scanButton, "S");

UxPutRowColumnType(filePane, "menu_pulldown");
UxCreateWidget(fileFrame_b3);

UxPutLabelString(fileFrame_b3, "");
UxPutMnemonic(fileFrame_b3, "");
UxCreateWidget(fileFrame_b2);

UxPutLabelString(fileFrame_b2, "");
UxPutMnemonic(fileFrame_b2, "");
UxCreateWidget(fileFrame_b1);

UxPutLabelString(fileFrame_b1, "");
UxPutMnemonic(fileFrame_b1, "");
UxCreateWidget(openButton);

UxPutBorderWidth(pulldownMenu, (int)0);
UxPutMnemonic(pulldownMenu, 'menu_bar');

UxPutRowColumnType(filePane, 'menu_pulldown');

UxPutLabelString(scanButton, 'Scan ...');
UxPutMnemonic(scanButton, 'S');

UxPutLabelString(openButton, 'Open ...');
UxPutMnemonic(openButton, 'O');

UxPutLabelString(viewButton, 'View ...');
UxPutMnemonic(viewButton, 'V');

UxPutLabelString(exitButton, 'Exit');
UxPutMnemonic(exitButton, 'E');

UxPutRowColumnType(runPane, 'menu_pulldown');

UxPutLabelString(runButton, 'Begin block selection process ...');
UxPutMnemonic(runButton, 'B');

UxPutRowColumnType(helpPane, 'menu_pulldown');

UxPutLabelString(colorRef, 'Color References...');

UxPutSubMenuId(fileCascade, 'filePane');
UxCreateWidget(fileCascade);

UxPutSubMenuId(runCascade, "runPane");
UxCreateWidget(runCascade);

UxPutSubMenuId(helpCascade, "helpPane");
UxCreateWidget(helpCascade);

UxPutSubMenuId(pulldownMenu, "helpCascade");
UxCreateWidget(pulldownMenu);

UxPutMenuHelpWidget(pulldownMenu, "helpCascade");

UxRealizeInterface(BlockSel);
UxWinDowsDoAddArea(mainWindow, pulldownMenu, NULL_SHIDGET,
        NULL_SHIDGET, NULL_SHIDGET, BlockSelBB);

return(BlockSel);
}

static void init_BlockSel(_UxCmsv)
    swidget     _Uxmsv;
{
    _UxCrossContext((_UxCBlockSel *) UxGetContext(_Uxmsv));

    UxPutX(BlockSel, (int)150);
    UxPutY(BlockSel, (int)10);
    UxPutWidth(BlockSel, (int)400);
    UxPutHeight(BlockSel, (int)100);
    UxPutTitle(BlockSel, "OCR Block Selection Pane");

    UxPutX(mainWindow, (int)100);
    UxPutY(mainWindow, (int)170);
    UxPutWidth(mainWindow, (int)300);
    UxPutHeight(mainWindow, (int)100);

    UxPutWidth(BlockSelBB, (int)90);
    UxPutHeight(BlockSelBB, (int)50);

    UxPutX(README, (int)7);
    UxPutY(README, (int)11);
    UxPutWidth(README, (int)124);
    UxPutHeight(README, (int)104);
    UxPutLabelString(README, "Block Selection Panel\n\nKanon Information Systems
        \nCopyright 1992 -- Version 0.1\nSkip-Pean Mangis");
```

【図39】

```
UxPutMnemonic(toolcolorsf, "C");

UxPutLabelString(fileCascade, "File");
UxPutMnemonic(fileCascade, "F");

UxPutLabelString(runCascade, "Run");
UxPutMnemonic(runCascade, "R");

UxPutLabelString(helpCascade, "Help");
UxPutMnemonic(helpCascade, "H");

_UxCreateContext(_UxCBlocks1 *) UxGetContext(_UxNew);

static void _UxCmContext(_UxCm)
    _UxCBlocks1    *_UxCm;
{
    _UxCm->Block1 = Block1;
    _UxCm->mainWindow = mainWindow;
    _UxCm->blockWidth = BlockWidth;
    _UxCm->blockHeight = BlockHeight;
    _UxCm->fileMenu = fileMenu;
    _UxCm->fileMenu_p1 = fileMenu_p1;
    _UxCm->fileMenu_p1_b1 = fileMenu_p1_b1;
    _UxCm->openButton = openButton;
    _UxCm->fileMenu_p1_b2 = fileMenu_p1_b2;
    _UxCm->viewButton = viewButton;
    _UxCm->fileMenu_p1_b3 = fileMenu_p1_b3;
    _UxCm->editButton = editButton;
    _UxCm->runPane = runPane;
    _UxCm->runButton = runButton;
    _UxCm->helpPane = helpPane;
    _UxCm->colorSelectionBox = colorSelectionBox;
    _UxCm->fileCascade = fileCascade;
    _UxCm->runCascade = runCascade;
    _UxCm->helpCascade = helpCascade;
}

static void _UxFromContext(_UxCm)
    _UxCBlocks1    *_UxCm;
{
    Block1 = _UxCm->Block1;
    mainWindow = _UxCm->mainWindow;
    BlockWidth = _UxCm->blockWidth;
    BlockHeight = _UxCm->blockHeight;
    fileMenu = _UxCm->fileMenu;
    openButton = _UxCm->openButton;
    fileMenu_p1 = _UxCm->fileMenu_p1;
    viewButton = _UxCm->viewButton;
    fileMenu_p1_b2 = _UxCm->fileMenu_p1_b2;
    editButton = _UxCm->editButton;
    runPane = _UxCm->runPane;
    runButton = _UxCm->runButton;
    helpPane = _UxCm->helpPane;
    colorSelectionBox = _UxCm->colorSelectionBox;
    fileCascade = _UxCm->fileCascade;
    runCascade = _UxCm->runCascade;
    helpCascade = _UxCm->helpCascade;
}

/* ARGSUSED */
static void activateCallback_openButton(UxWidget, UxContext, UxCallbackArg)
    Widget          UxWidget;
    XtPointer       UxContext;
    XtPointer       UxCallbackArg;
{
    _UxCBlocks1 *UxSaveCtx, *UxContext;
    Widget          widget = UxWidget;

    UxSaveCtx = UxContext =
        (_UxCBlocks1 *) UxGetContext(UxThisWidget(widget));

    {
        handle scanner_handle;
        scanner_handle = UxCreateSubproc ("/usr/bin/testtest", NULL, UXAppendCh);

        if (scanner_handle == -1) {
            error_create_subproc ();
            return;
        }
        if (UxRunSubproc (scanner_handle, NULL) == -1) {
            error_run_subproc ();
            return;
        }
    }
    _UxCmContext = UxSaveCtx;
}

/* ARGSUSED */
static void activateCallback_viewButton(UxWidget, UxContext, UxCallbackArg)
    Widget          UxWidget;
    XtPointer       UxContext;
    XtPointer       UxCallbackArg;
{
    _UxCBlocks1 *UxSaveCtx, *UxContext;
    Widget          widget = UxWidget;

    UxSaveCtx = UxContext =
        (_UxCBlocks1 *) UxGetContext(UxThisWidget(widget));

    {
        handle view_handle;
        view_handle = UxCreateSubproc ("/usr/local/bin/X11/imageview", NULL, NULL
        perror();
        if (view_handle == -1) {
            error_create_subproc ();
            return;
        }
        if (UxRunSubproc (view_handle, NULL) == -1) {
```

【図４０】

```
        error_run_subproc ();
        return;
    }
}

_UxTxContext(UxContext);

}

/* ARGSUSED */
static void activateCallback_exitButton(UxWidget, UxContext, UxCallbackArg)
    Widget      UxWidget;
    _UxCBlockRel    *UxContext;
    int         UxCallbackArg;
{

    swidget     UxThisWidget;

    UxThisWidget= UxWidgetToSwidget(UxWidget);
    _UxFromContext(UxContext);

    {
        UxPopupInterface(QuitDialog, no_grab);
    }

    _UxToContext(UxContext);

}

/* ARGSUSED */
static void activateCallback_pushbutton1(UxWidget, UxContext, UxCallbackArg)
    Widget      UxWidget;
    _UxCBlockRel    *UxContext;
    int         UxCallbackArg;
{

    swidget     UxThisWidget;
    char        *foreground, *background;
    int         height, width;

    UxThisWidget= UxWidgetToSwidget(UxWidget);
    _UxFromContext(UxContext);

    {
        /* Start execution of your program here */
        UxPopupInterface(DisplayForm, no_grab);

        height = UxGetHeight (DisplayForm_DrawArea);

/*      printf ("Height => %d\n", height);
*/
        width = UxGetWidth(DisplayForm_DrawArea);

/*      printf ("Width => %d\n", width);
*/

        foreground = UxGetForeground(DisplayForm_DrawArea);
        background = UxGetBackground(DisplayForm_DrawArea);
/*      printf("f = %s b = %s \n", foreground, background);
*/

        Block_process(UxGetWidget(DisplayForm_DrawArea), width, height);
    }

    _UxToContext(UxContext);

}

/* ARGSUSED */
static void activateCallback_colorRef(UxWidget, UxContext, UxCallbackArg)
    Widget      UxWidget;
    _UxCBlockRel    *UxContext;
    int         UxCallbackArg;
{

    swidget     UxThisWidget;
    int         width, height;
    char        *foreground, *background;
    char        *font;

    UxThisWidget= UxWidgetToSwidget(UxWidget);
    _UxFromContext(UxContext);

    {
        UxPopupInterface(InfoForm, no_grab);
        height = UxGetHeight (InfoFormDrawArea);
        printf ("Height => %d\n", height);
        width = UxGetWidth(InfoFormDrawArea);
        printf ("Width => %d\n", width);

        foreground = UxGetForeground(InfoFormDrawArea);
        background = UxGetBackground(InfoFormDrawArea);
/*      printf("f = %s b = %s \n", foreground, background);
*/
        font = UxGetFontList(InfoFormDrawArea);
/*      printf("font = %s \n", font);
*/

        color_reference(UxGetWidget(InfoFormDrawArea), width, height);
    }

    _UxToContext(UxContext);

}
```

【図41】

```
/*****************************************************************/
/*                                                               */
/*          This file was generated by Architect                 */
/*                                                               */
/*          You are free to modify or replace this header.       */
/*                                                               */
/*****************************************************************/
/*  Creation: Wed Mar 31 20:32:43 EST 1993                       */
/*****************************************************************/
#include <stdio.h>
#include "UxLib.h"
#include "UxFrBox.h"
#include "UxCallbk.h"
#include "UxCbProc.h"
#include "block.h"
static XmStringCharSet charset = (XmStringCharSet) XmSTRING_DEFAULT_CHARSET;
char filename[FILE_LENGTH];

typedef struct
        {
        swidget FileSelectionDialog;
                FileSelectionBox;
                swidget parent;
        } _UxCFileSelectionDialog;

static void     _UxCreateContext(), _UxFreeContext();

static void     cancelCallback_FileSelectionBox(), okCallback_FileSelectionBox();

static swidget  build_FileSelectionDialog();
static swidget  _UxCreate_FileSelectionDialog();
static void     init_FileSelectionDialog();

swidget         FileSelectionDialog;
swidget         FileSelectionBox;

extern void open_diff_file();

static swidget  parent;

char *get_input_filename ()
{
    int n = 0;
    Arg args[10];
    XmString str        = NULL;
    char *name          = NULL;

    XtSetArg (args[n], XmNdirectory, &str); n++;
    XtGetValues (UxGetWidget(FileSelectionBox), args, n);
    XmStringGetLtoR (str, charset, &name);
    return name;
}

void OutputHandler (fd, data)
    int fd;
    char *data;
{
    printf ("In output handler\n");
}

swidget create_FileSelectionDialog(_Uxparent)
        swidget _Uxparent;
        {
                swidget _Uxrtrn;
                _UxCFileSelectionDialog *_UxCon = _UxCreate_FileSelectionDialog(_Uxparent);

    if(UxIsValidWidget(FileSelectionDialog))
        _UxCtxToCon((_UxCFileSelectionDialog *) UxGetContext(FileSelectionDialog));

    return(_Uxrtrn);
    }

static swidget _Uxcreate_FileSelectionDialog(_Uxparent)
        swidget _Uxparent;
        {
        swidget rtrn;
        _UxCFileSelectionDialog *_UxCon;

        parent = _Uxparent;
        rtrn = _Build_FileSelectionDialog();

        XtManageChild(XmFileSelectionBoxGetChild(UxGetWidget(FileSelectionBox), XmDIALOG_APPLY_BUTTON));
        return(rtrn);
        }

static swidget build_FileSelectionDialog()
{
        _UxCFileSelectionDialog *_UxCon;

        _UxCon = (_UxCFileSelectionDialog *) XtMalloc(sizeof(_UxCFileSelectionDialog));

        _UxCon->parent = parent;

        FileSelectionDialog = UxCreateDialogShell("FileSelectionDialog", parent);
        UxPutContext(FileSelectionDialog, _UxCon);
        FileSelectionBox = UxCreateFileSelectionBox("FileSelectionBox", FileSelectionDialog);

        _UxCon->Context = UxNoContext;

        init_FileSelectionDialog(FileSelectionDialog);

        UxCreateWidget(FileSelectionDialog);

        UxCreateWidget(FileSelectionBox);
        UxAddCallback(FileSelectionBox, "cancelCallback", cancelCallback_FileSelectionBox, _UxCon);
        UxAddCallback(FileSelectionBox, "okCallback", okCallback_FileSelectionBox, _UxCon);

        UxRealizeInterface(FileSelectionDialog);

        return(FileSelectionDialog);
}

static void init_FileSelectionDialog(_Uxsw)
        swidget _Uxsw;
{
        _UxCFileSelectionDialog *_UxCon = (_UxCFileSelectionDialog *) UxGetContext(_Uxsw);

        UxPutTitle(FileSelectionDialog, "File Selection");

        XtVaSetValues(FileSelectionDialog, XmNdialogStyle, XmDIALOG_PRIMARY_APPLICATION_MODAL, NULL);

        _UxCFileSelectionDialog *_UxCon = (_UxCFileSelectionDialog *) UxGetContext(_Uxsw);
}

static void _UxCtxToCon(_UxCon)
        _UxCFileSelectionDialog *_UxCon;
{
```

【図42】

```
        _UxCon->FileSelectionDialog= FileSelectionDialog;
        _UxCon->FileSelectionBox= FileSelectionBox;

        _UxCon->parent= parent;
}

static void _UxFromContext(_UxCon)
        _UxCFileSelectionDialog *_UxCon;
{
        FileSelectionDialog= _UxCon->FileSelectionDialog;
        FileSelectionBox= _UxCon->FileSelectionBox;

        parent= _UxCon->parent;
}

/* ARGSUSED */
static void cancelCallback_FileSelectionBox(UxWidget, UxContext, UxCallbackArg)
        Widget          UxWidget;
        _UxCFileSelectionDialog *UxContext;
        int             UxCallbackArg;
{
        swidget         UxThisWidget;

        UxThisWidget= UxWidgetToSwidget(UxWidget);
        _UxFromContext(UxContext);

        {
            UxPopdownInterface(FileSelectionDialog);
        }

        _UxToContext(UxContext);
}

/* ARGSUSED */
static void okCallback_FileSelectionBox(UxWidget, UxContext, UxCallbackArg)
        Widget          UxWidget;
        _UxCFileSelectionDialog *UxContext;
        int             UxCallbackArg;
{
        swidget         UxThisWidget;

        UxThisWidget= UxWidgetToSwidget(UxWidget);
        _UxFromContext(UxContext);

        {
            handle tiff_handle;
            UxPopdownInterface(FileSelectionDialog);
            strcpy (filename, get_input_filename ());
            printf ("Open [%s]\n", filename);
            tiff_handle = UxCreateSubproc ("/usr/local/bin/X11/imageview", (filename,
UxAppendPe);
            if (tiff_handle == -1) {
                error_create_subproc ();
                return;
            }
            if (UxRunSubproc (tiff_handle, NULL) == -1) {
                error_run_subproc ();
                return;
            }
            open_tiff_file(filename);
        }

        _UxToContext(UxContext);
}
```

【図48】

```
}

static void _UxFromContext(_UxCon)
        _UxCInfoForm    *_UxCon;
{
        InfoForm= _UxCon->InfoForm;
        InfoFormClose_PBG= _UxCon->InfoFormClose_PBG;
        InfoFormAreaG= _UxCon->InfoFormAreaG;
}

/* ARGSUSED */
static void activateCallback_InfoFormClose_PBG(UxWidget, UxContext, UxCallbackArg)
        Widget          UxWidget;
        _UxCInfoForm    *UxContext;
        int             UxCallbackArg;
{
        swidget         UxThisWidget;

        UxThisWidget= UxWidgetToSwidget(UxWidget);
        _UxFromContext(UxContext);

        {
            UxPopdownInterface (InfoForm);
        }

        _UxToContext(UxContext);
}
```

【図43】

```
/*****************************************************************/
/*                                                               */
/*          This file was generated by Architect                 */
/*                                                               */
/*       You are free to modify or replace this header           */
/*                                                               */
/*****************************************************************/
/*    Creation: Wed Mar 25 20:37:32 EST 1992                     */
/*****************************************************************/
typedef struct
{
    Widget QuitDialog;
    Widget QuitMessageBox;
    Widget parent;
} _UxCQuitDialog;

#include <stdio.h>
#include <Xm/Xm.h>
#include <Xm/MessageBox.h>
#include <Xm/MwmUtil.h>
#include <X11/MwmUtil.h>

static _UxCQuitDialog    *_UxCContext;

static void    _UxCreateContext();

static Widget  build_QuitDialog();
static Widget  _Uxcreate_QuitDialog();
static void    init_QuitDialog();

Widget  QuitDialog;
static Widget  QuitMessageBox;

static Widget  parent;

Widget create_QuitDialog(_Uxparent)
    Widget _Uxparent;
{
    Widget    _Uxrtrn;
    _Uxparent = _Uxcreate_QuitDialog(_Uxparent);
    if(UxValidGdget(QuitDialog))
        _UxCContext->QuitDialog = UxGetContext(QuitDialog);
    return(_Uxrtrn);
}

static Widget _Uxcreate_QuitDialog(_Uxparent)
    Widget _Uxparent;
{
    Widget rtrn;

    parent = _Uxparent;
    rtrn = build_QuitDialog();

    XtManageChild (XmMessageBoxGetChild(UxGetWidget(QuitMessageBox),
                   XmDIALOG_HELP_BUTTON));
    return(rtrn);
}

static Widget build_QuitDialog()
{
    _UxCQuitDialog *_UxCon;

    _UxCon= (_UxCQuitDialog *) UxMalloc(sizeof(_UxCQuitDialog));

    _UxCon->parent= parent;

    QuitDialog= UxCreateDialogShell("QuitDialog", parent);
    UxPutContext(QuitDialog, _UxCon);
    QuitMessageBox= UxCreateMessageBox("QuitMessageBox", QuitDialog);

    _UxCContext=_UxCon;

    init_QuitDialog(QuitDialog);

    UxCreateWidget(QuitDialog);

    UxCreateWidget(QuitMessageBox);
    UxAddCallback(QuitMessageBox, "okCallback", okCallback_QuitMessageBox, _UxCon
    );

    XmDoLayoutUpdate(QuitDialog);
    UxOverrideGeometries(QuitDialog, (Int)MWM_DECOR_BORDER | MWM_DECOR_TITLE | MWM
_DECOR_MENU);
    UxMwmFunctions(QuitDialog, (Int)MWM_FUNC_MOVE | MWM_FUNC_CLOSE);
    XmUpdate(QuitDialog);

    UxRealizeInterface(QuitDialog);

    return(QuitDialog);
}

static void init_QuitDialog(_Uxnew)
    Widget _Uxnew;
{
    _UxFromContext(_UxCQuitDialog *) UxGetContext(_Uxnew);

    UxPutX(QuitDialog, (int)490);
    UxPutY(QuitDialog, (int)410);
    UxPutWidth(QuitDialog, (int)170);
    UxPutHeight(QuitDialog, (int)120);

    UxPutDialogStyle(QuitMessageBox, "dialog_primary_application_modal");
    UxPutDialogTitle(QuitMessageBox, "Quit Dialog");
    UxPutDialogType(QuitMessageBox, "dialog_question");
    UxPutMessageString(QuitMessageBox, "Quit Block Selection Demo?");

    _UxCContext=(_UxCQuitDialog *) UxGetContext(_Uxnew);
}

static void _UxCreateContext(_UxCon)
    _UxCQuitDialog *_UxCon;
{
    _UxCon->QuitDialog= QuitDialog;
    _UxCon->QuitMessageBox= QuitMessageBox;

    _UxCon->parent= parent;
}

static void _UxFromContext(_UxCon)
    _UxCQuitDialog *_UxCon;
{
    QuitDialog= _UxCon->QuitDialog;
    QuitMessageBox= _UxCon->QuitMessageBox;

    parent= _UxCon->parent;
}
```

【図45】

```
/*****************************************************************************/
/*                                                                           */
/*          This file was generated by Architect                             */
/*                                                                           */
/*          You are free to modify or replace this header.                   */
/*                                                                           */
/*****************************************************************************/
/*    Creation: Wed Mar 23 20:27:48 EST 1991                                 */
/*****************************************************************************/
#include <stdio.h>
#include "Xlib.h"
#include "DXmCSpin.h"
#include "DXmPush.h"
#include "myform.h"

typedef struct
{
    Widget DisplayForm,
           DisplayFormClose_PB,
           DisplayForm_DrawArea;
} _UxCDisplayForm;

static void    _UxFreeContext(), _UxFromContext();

static void    activateCallback_DisplayFormClose_PB();

static Widget  build_DisplayForm();
static Widget  _Uxcreate_DisplayForm();
static void    init_DisplayForm();

Widget  DisplayForm;
static Widget  DisplayFormClose_PB;
Widget         DisplayForm_DrawArea;


Widget create_DisplayForm()
{
    Widget        _UxRtrn;
    _UxRtrn = _Uxcreate_DisplayForm();

    if(!UxIsValidWidget(DisplayForm))
        _UxFromContext((_UxCDisplayForm *) UxGetContext(DisplayForm));
    return(_UxRtrn);
}

static Widget _Uxcreate_DisplayForm()
{
    Widget rtrn;

    rtrn = build_DisplayForm();

    return(rtrn);
}

static Widget build_DisplayForm()
{
    _UxCDisplayForm *_UxCon;

    _UxCon = (_UxCDisplayForm *) UxAllocClosure(_UxCDisplayForm);

    DisplayForm= UxCreateForm("DisplayForm", NO_PARENT);
    UxPutContext(DisplayForm, _UxCon);
    UxPutDefaultShell(DisplayForm, "topLevelShell");
    DisplayFormClose_PB= UxCreatePushButtonGadget("DisplayFormClose_PB", DisplayForm);
yForm);
    DisplayForm_DrawArea= UxCreateDrawingArea("DisplayForm_DrawArea", DisplayForm
);

    _UxCon->DisplayForm = DisplayForm;
    _UxCon->DisplayFormClose_PB = DisplayFormClose_PB;
    _UxCon->DisplayForm_DrawArea = DisplayForm_DrawArea;

    UxPutContext((_UxCDisplayForm *) UxGetContext(_UxCon),
                 _UxCon);

    UxPutBottomAttachment(DisplayForm_DrawArea, "attach_form");
    UxPutBottomOffset(DisplayFormClose_PB, (int)5);
    UxPutLeftAttachment(DisplayFormClose_PB, "attach_form");
    UxPutLeftOffset(DisplayFormClose_PB, (int)5);
    UxCreateWidget(DisplayFormClose_PB);
    UxAddCallback(DisplayFormClose_PB, "activateCallback", activateCallback_Disp
layFormClose_PB, _UxCon);

    UxPutBottomAttachment(DisplayForm_DrawArea, "attach_widget");
    UxPutBottomOffset(DisplayForm_DrawArea, (int)5);
    UxPutBottomWidget(DisplayForm_DrawArea, "DisplayFormClose_PB");
    UxPutLeftAttachment(DisplayForm_DrawArea, "attach_form");
    UxPutLeftOffset(DisplayForm_DrawArea, (int)5);
    UxPutRightAttachment(DisplayForm_DrawArea, "attach_form");
    UxPutRightOffset(DisplayForm_DrawArea, (int)5);
    UxPutTopAttachment(DisplayForm_DrawArea, "attach_form");
    UxPutTopOffset(DisplayForm_DrawArea, (int)5);
    UxCreateWidget(DisplayForm_DrawArea);


    UxRealizeInterface(DisplayForm);

    return(DisplayForm);
}

static void init_DisplayForm(_Uxrw)
    Widget        _Uxrw;
{
    _UxFromContext((_UxCDisplayForm *) UxGetContext(_Uxrw));

    UxPutX(DisplayForm, (int)200);
    UxPutY(DisplayForm, (int)400);
    UxPutWidth(DisplayForm, (int)700);
    UxPutHeight(DisplayForm, (int)570);
    UxPutWidth(DisplayFormClose_PB, (int)90);
    UxPutHeight(DisplayFormClose_PB, (int)30);
    UxPutLabelString(DisplayFormClose_PB, "close");
    UxPutMnemonic(DisplayFormClose_PB, "c");
    UxPutRecomputeSize(DisplayFormClose_PB, "false");

    UxPutX(DisplayForm_DrawArea, (int)10);
    UxPutY(DisplayForm_DrawArea, (int)90);
    UxPutWidth(DisplayForm_DrawArea, (int)680);
    UxPutHeight(DisplayForm_DrawArea, (int)510);
    UxPutResizePolicy(DisplayForm_DrawArea, "resize_none");
}
```

【図47】

```
/*****************************************************************************
/***                                                                       ***/
/***          This file was generated by Architect                         ***/
/***                                                                       ***/
/***      You are free to modify or replace this header.                   ***/
/***                                                                       ***/
/*****************************************************************************
/***    Creation: Fri Mar 27 16:36:57 JST 1992                             ***/
/*****************************************************************************/
#include <stdio.h>
#include "UxLib.h"
#include "UxUtils.h"
#include "UxXmMsg.h"
#include "UxForm.h"

typedef struct
{
    Widget  InfoForm;
            InfoFormClose_PBG;
            InfoFormDrawA;
} _UxCInfoForm;

static void    _UxCreateContext(), _UxFreeContext();

static void    ActivateCallback_InfoFormClose_PBG();

static void    build_InfoForm();
static Widget  _UxCreate_InfoForm();
static void    init_InfoForm();

Widget
static Widget  InfoForm;
static Widget  InfoFormClose_PBG;
Widget InfoFormDraw;

Widget create_InfoForm()
{
    Widget        _UxRtrn;
    _UxCInfoForm *_UxCtx;
    Widget        rtrn;

    if(UxIsValidWidget(InfoForm))
        _UxCreateContext((_UxCInfoForm *) UxGetContext(InfoForm));
    return(_UxRtrn);
}

static Widget _UxCreate_InfoForm()
{
    Widget rtrn;

    rtrn = build_InfoForm();

    return(rtrn);
}

static Widget build_InfoForm()
{
    _UxCInfoForm    *_UxCtx;

    _UxCtx = (_UxCInfoForm *) UxAllocate(sizeof(_UxCInfoForm));

    InfoForm= UxCreateForm("InfoForm", NO_PARENT);
    UxPutContext(InfoForm, _UxCtx);
    UxPutDefaultWidget(InfoForm, "topLevelShell");
    InfoFormClose_PBG= UxCreatePushButtonGadget("InfoFormClose_PBG", InfoForm);

    InfoFormDraw= UxCreateDrawingArea("InfoFormDraw", InfoForm);

    _UxCtx->InfoForm= InfoForm;
    _UxCtx->InfoFormClose_PBG= InfoFormClose_PBG;
    _UxCtx->InfoFormDraw= InfoFormDraw;

    init_InfoForm(InfoForm);

    UxCreateWidget(InfoForm);

    UxPutBottomAttachment(InfoFormClose_PBG, "attach_form");
    UxPutBottomOffset(InfoFormClose_PBG, (int)1);
    UxPutLeftAttachment(InfoFormClose_PBG, "attach_none");
    UxPutLeftOffset(InfoFormClose_PBG, (int)2);
    UxPutRightAttachment(InfoFormClose_PBG, "attach_none");
    UxCreateWidget(InfoFormClose_PBG);
    UxAddCallback(InfoFormClose_PBG, activateCallback, activateCallBACK_InfoFormClose_PBG, _UxCtx);

    UxPutBottomAttachment(InfoFormDraw, "attach_widget");
    UxPutBottomOffset(InfoFormDraw, (int)2);
    UxPutBottomWidget(InfoFormDraw, "InfoFormClose_PBG");
    UxPutLeftAttachment(InfoFormDraw, "attach_form");
    UxPutLeftOffset(InfoFormDraw, (int)2);
    UxPutRightOffset(InfoFormDraw, (int)2);
    UxPutRightAttachment(InfoFormDraw, "attach_form");
    UxPutTopAttachment(InfoFormDraw, "attach_form");
    UxPutTopOffset(InfoFormDraw, (int)2);
    UxCreateWidget(InfoFormDraw);

    UxResolveInterface(InfoForm);

    return(InfoForm);
}

static void init_InfoForm(_Uxw)
    Widget  _Uxw;
{
    _UxCInfoForm *_UxCtx = (_UxCInfoForm *) UxGetContext(_Uxw);

    UxPutX(InfoForm, (int)780);
    UxPutY(InfoForm, (int)0);
    UxPutWidth(InfoForm, (int)360);
    UxPutHeight(InfoForm, (int)390);
    UxPutDialogTitle(InfoForm, "Block Selection Demo -- Help");

    UxPutX(InfoFormClose_PBG, (int)10);
    UxPutY(InfoFormClose_PBG, (int)200);
    UxPutWidth(InfoFormClose_PBG, (int)80);
    UxPutHeight(InfoFormClose_PBG, (int)30);
    UxPutLabelString(InfoFormClose_PBG, "Done");

    UxPutX(InfoFormDraw, (int)10);
    UxPutY(InfoFormDraw, (int)10);
    UxPutWidth(InfoFormDraw, (int)340);
    UxPutHeight(InfoFormDraw, (int)180);
    UxPutBackground(InfoFormDraw, "black");
    UxPutResizePolicy(InfoFormDraw, "resize_none");

    _UxCreateContext((_UxCInfoForm *) UxGetContext(_Uxw));
}

static void _UxCreateContext(_UxCInfoContext)
    _UxCInfoContext *_UxCon;
{
    InfoForm= _UxCon->InfoForm;
    InfoFormClose_PBG= _UxCon->InfoFormClose_PBG;
    InfoFormDraw= _UxCon->InfoFormDraw;
```

【図49】

```
#include <stdio.h>
#include <stdlib.h>
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <Xm/Xm.h>
#include <Xm/DrawingA.h>
#include <X11/Xutil.h>
#include <time.h>
#include "block.h"

struct timeval {first, second, lapsed;
struct timezone* tzp;

/*****************************************************************
 * Start the timing clock
 *****************************************************************/
start_clock ()
{
  gettimeofday (&first, &tzp);
}

/*****************************************************************
 * Stop the timing clock and return the elapsed time in milliseconds.
 *****************************************************************/
double stop_clock ()
{
  double time;

  gettimeofday (&second, &tzp);
  if (first.tv_usec > second.tv_usec) {
      second.tv_usec += 1000000;
      second.tv_sec--;
    }
  lapsed.tv_usec = second.tv_usec - first.tv_usec;
  lapsed.tv_sec = second.tv_sec - first.tv_sec;
  time = (double)lapsed.tv_usec / (double)1000;
  printf ("***** Elapsed time = %.3f\n msec.", time);
  return time;
}

/*****************************************************************/
/* Return the basename of a file without the directory */
/*****************************************************************/
char *extract_base_file_no_dir (name)
  char *name;
{
  char tmp[FILE_LENGTH], *pos;

  strcpy (tmp, name);
  if ((pos = strrchr(tmp, '/')) != NULL)
      return pos+1;
  else
      return name;
}

/*****************************************************************/
/* Return the basename of a file */
/*****************************************************************/
char *extract_base_file (name)
  char *name;
{
  char tmp[FILE_LENGTH], *pos;

  strcpy (tmp, name);
  if ((pos = strchr(tmp, '.')) != NULL)
      *pos = '\0';
  return tmp;
}

/*****************************************/
/* Check if file exists open.*/
/*****************************************/
int check_file (name)
  char *name;
{
  FILE *fp;

  if ((fp = fopen (name, "r")) == NULL)
      return 0;
  fclose (fp);
  return 1;
}

print_error (mesg)
  char *mesg;
{
  printf ("****Error: %s\n", mesg);
}

error_create_subproc ()
{
  printf ("****Error: Cannot create a subprocess!\n");
}

error_subproc_exit_callback ()
{
  printf ("****Error: Cannot set subprocess exit callback!\n");
}

error_run_subproc ()
{
  printf ("****Error: Cannot start a subprocess!\n");
}
```

【図50】

```
/*****************************************************************/
/*                                                               */
/*                  File: xm.c                                   */
/*                  Author: Shih-Yuan Wang                       */
/*                  Date : 3-20-92                               */
/*                  Copyright 1992 Canon Information Systems     */
/*                                                               */
/*****************************************************************/

/* ------------------------Includes------------------------ */
#include <stdio.h>
#include <X11/Xlib.h>
#include "color.h"

/* ------------------------Global Var---------------------- */

extern Widget TmpTopLevel;
GC drawing_gc;
XGCValues gcv;

void init_graphics(wd)
Widget wd;
{
    drawing_gc = XCreateGC(XtDisplay(wd),
                           DefaultRootWindow(XtDisplay(wd)),
                           GCFunction, &gcv);
}

int xm_get_colorized(red, green, blue, wd)
int red, green, blue;
Widget wd;
{
  XColor color;
  Colormap cmap;

  cmap = DefaultColormap(XtDisplay(wd),
                         DefaultScreen(XtDisplay(wd)));

  color.red = red*256;
  color.green = green*256;
  color.blue = blue*256;

  if (XAllocColor(XtDisplay(wd), cmap, &color))
    return (color.pixel);
  printf("Warning: couldn't allocate requested color\n");
  return BlackPixel(XtDisplay(wd), DefaultScreen(XtDisplay(wd)));
}

void xm_set_foreground(color, wd)
int color;
Widget wd;
{
    XSetForeground(XtDisplay(wd), drawing_gc,
                   xm_get_color(color_cmap[color][R],
                                color_cmap[color][G],
                                color_cmap[color][B],
                                wd));
}

void xm_plot_xy(wd, x, y)
Widget wd;
int x, y;
{
    XDrawPoint(XtDisplay(wd),XtWindow(wd), drawing_gc, x, y);
}

xm_unplot_xy(wd, x, y)
Widget wd;
int x, y;
{
    /* Flush the queue to X-server */
    XFlush(XtDisplay(wd));
}

xm_flt_win(wd)
Widget wd;
{
    XClearArea(XtDisplay(wd),XtWindow(wd),0,0,0,0,TRUE);
}

xm_draw_box(wd, x,y,width,height)
Widget wd;
int   x, y, width, height;
{
    XDrawRectangle(XtDisplay(wd), XtWindow(wd), drawing_gc, x,y,width,height);
}

xm_string(wd, x,y, str)
Widget wd;
int x, y;
char *str;
{
    XDrawString(XtDisplay(wd), XtWindow(wd), drawing_gc, x, y,str,strlen(str));
}

xm_win_dim(wd, width,height)
Widget wd;
int *width;
int *height;
{
    Arg args[2];
    Dimension win_width,win_height;

    XtSetArg(args[0],XtNwidth,&win_width);
    XtSetArg(args[1],XtNheight,&win_height);
    XtGetValues(wd,args,2);

    printf("width = %d length = %d\n", win_width, win_height);
    *width = win_width;
```

【図５１】

```
    *height = win_height;
}

xs_draw_line(wd, x1,y1,x2,y2)
Widget wd;
int x1, y1, x2, y2;
{
    XDrawLine(XtDisplay(wd), XtWindow(wd), drawing_gc,  x1, y1, x2, y2);
}

xs_color_point(wd, x, y, color)
Widget wd;
int x, y, color;
{
    Arg args[1];
    int foreground;

    XtSetArg(args[0],XtNforeground,&foreground);
    XtGetValues(wd,args,1);

    XSetForeground(XtDisplay(wd), drawing_gc, color);
    XDrawPoint(XtDisplay(wd), XtWindow(wd), drawing_gc, x, y);
    XSetForeground(XtDisplay(wd), drawing_gc, foreground);
}

xs_color_line(wd, x1, y1, x2, y2, color)
Widget wd;
int x1, y1, x2, y2, color;
{
    Arg args[1];
    int foreground;

    XtSetArg(args[0],XtNforeground,&foreground);
    XtGetValues(wd,args,1);

    XSetForeground(XtDisplay(wd), drawing_gc, color);
    XDrawLine(XtDisplay(wd), XtWindow(wd), drawing_gc, x1, y1, x2, y2);
    XSetForeground(XtDisplay(wd), drawing_gc, foreground);
}

xs_color_box(wd, x1, y1, x2, y2,  color)
Widget wd;
int x1, y1, x2, y2, color;
{
    Arg args[1];
    int foreground;

    XtSetArg(args[0],XtNforeground,&foreground);
    XtGetValues(wd,args,1);

    XSetForeground(XtDisplay(wd), drawing_gc, color);
    XDrawRectangle(XtDisplay(wd), XtWindow(wd), drawing_gc, x1, y1, x2, y2);
    XSetForeground(XtDisplay(wd), drawing_gc, foreground);
}

xs_color_str(wd, x, y, str, color)
Widget wd;
int x, y, color;
char *str;
{
    Arg args[1];
    int foreground;

    XtSetArg(args[0],XtNforeground,&foreground);
    XtGetValues(wd,args,1);

    XSetForeground(XtDisplay(wd), drawing_gc, color);
    XDrawString(XtDisplay(wd), XtWindow(wd), drawing_gc, x, y, str, strlen(str));
    XSetForeground(XtDisplay(wd), drawing_gc, foreground);
}

static char **font;
XA_def_font(wd)
Widget wd;
{
    int i, num = 50;

    font = XListFonts(XtDisplay(wd), "*", 50, &num);

    printf("num = %d \n", num);
}

xa_set_font(wd, fid)
Widget wd;
int fid;
{

    XFontStruct *font_info;

    if ((font_info = XLoadQueryFont(XtDisplay(wd), font[fid]))==NULL){
        printf("can not open the font fid = %d \n", fid);
        exit(-1);
    }

    XSetFont(XtDisplay(wd), drawing_gc, font_info->fid);
    printf("fid = %d \n", fid);
}
```

【図52】

```
/*************************************************************************/
/*************     Program: cimages.c                          ***********/
/*************     Author:  Shin-Yuan Wang, Chris Sherrick     ***********/
/*************     Date:    3-10-92                            ***********/
/*************Copyright 1992 Canon Information Systems         ***********/
/*************************************************************************/
/*************************************************************************/


#include "blockgeneral.h"
#include "images.h"
#include <malloc.h>


extern unsigned char **cmatrix();


/*************************************************************************/
/**********   Allocate memory for an cimage from memory    ***************/
/*************************************************************************/

struct images *image_allocate(r, c)
int     r, c;
{

  struct images *page;

  page = (struct images *)malloc(sizeof(struct images));
  if (page == NULL) {
     printf("allocation error: in image_allocate\n");
     exit(1);
  }
  if (r <= 0 || c <= 0)
  {
    printf("Illegal cimage size: %d by %d \n", r ,c);
    exit(1);
  };
  page->sizeR = r;                              /* Set cimage sizes     */
  page->sizeC = c;
  page->pixel = cmatrix(0, page->sizeR-1, 0, page->sizeC-1);
  return page;
}


/*************************************************************************/
/*******     Deallocate memory for an cimage from memory   ***********/
/*************************************************************************/

void image_deallocate(page)
struct images *page;
{
  if ((page->sizeR != 0) && (page->sizeC != 0))
  {
    free_cmatrix(page->pixel, 0, page->sizeR-1, 0, page->sizeC-1);
                                               /* Zero out the sizes  */
    page->sizeR = 0;
    page->sizeC = 0;
  }
}

/*************************************************************************
*/
/*************** Read in a byte from a tiff file
*/
/*************************************************************************
*/

int getlbyte(f, file_position, eoferror)
FILE *f;
long int *file_position;
int  eoferror;
{
  int c1;
  c1 = getc(f);
  if ((c1 == EOF) && (eoferror == 1))
  {
    printf("\n Error: Premature End of File (in getlbyte())\n");
    exit(1);
  }
  (*file_position)++;
  return c1;
}

/*************************************************************************
                                                              ***********/
/* Read in an integer from a file.  (LSB, MSB)
long int get2bytes(f, fp)
FILE *f;
long int *fp;
{
  int c1, c2;
  long int c3;
  c1 = getlbyte(f, fp, 1);
  c2 = getlbyte(f, fp, 1);
  c3 = (long int)(c1 + c2*256);
  return  c3;
}

/*************************************************************************
                                                              ***********/
/*  Read in a long integer
long int get4bytes(f, fp)
FILE *f;
long int *fp;
{
  long int c1, c2;
  c1 = get2bytes(f, fp);
  c2 = get2bytes(f, fp);
  return c1 + c2*65536;
}


/*************************************************************************/
/*******      Read in part of a Tiff file                  ***********/
/*************************************************************************/

struct images *readtiff(filename)
char *filename;
{
  FILE *tagfile;                    /* Input file                          */
  long int tagfile_position;        /* The our file position (in bytes)   */
  long int width;                   /* Width of picture                    */
  long int length;                  /* Length of picture                   */
  long int black;                   /* Value of Black pixel                */
  long int bps;                     /* Bytes per strip                     */
  long int raster_offset;           /* Start of image data                 */
  long int numtags, tag, type, len, value;    /* Tag values            */
  long int i,j;
  int  all_length, all_width;
  int  v;
  struct images *page;

  if ((tagfile = fopen(filename, "rb")) == NULL)
  {
```

【図53】

```
  printf("\nError in opening %s \n ", filename);
  exit(1);
}
tagfile_position = 0;

/* Read in header and make sure it's Intel */
i = getbytes(tagfile, &tagfile_position);
/* printf("i = %x \n", i);*/
if (i != 0x4949)
{
  printf("... Error: TIFF file not in Intel Format\n");
  exit(1);
}

/* Read in Magic number                    */
i = getbytes(tagfile, &tagfile_position);
/* printf("i = %x \n", i); */
if (i != 0x002A)
{
  printf(" BAD Magic Number for TIFF file!!!\n");
  exit(1);
}

/* Find First File Directory               */
j = getbytes(tagfile, &tagfile_position);
while(tagfile_position < j)
{
  i = getbytes(tagfile, &tagfile_position);
}

/* Tags in Directory                       */
numtags = getbytes(tagfile, &tagfile_position);
/* printf("numtags = %x \n", numtags); */
while (--numtags != 0){
  tag = getbytes(tagfile, &tagfile_position);
  type = getbytes(tagfile, &tagfile_position);
  len = getbytes(tagfile, &tagfile_position);
  value = getbytes(tagfile, &tagfile_position);

  if ((len != 1) && (tag & 32768) == 0)
  {
    printf("Error: Tag Length != 1\n");
    exit(1);
  }
/* printf("tag = %x \t", tag); */
  switch (tag)
  {
    case 0x00FE:          /* NewSubFileType  */
      if (value != 0)
      {
        printf("Can't Deal with Non-Standard NewSubFileType\n");
        exit(1);
      }
      break;
    case 0x00FF:          /* NewSubFileType  */
      if (value != 1)
      {
        printf("Subfile Type not full resolution\n");
        exit(1);
      }
      break;
    case 0x0100:          /* Width           */
      width = value;
      break;
    case 0x0101:          /* Length          */
      length = value;
      break;
    case 0x0102:          /* Bits per sample */
      if (value != 1)
      {
        printf("; Error: Bits/Sample must be 1!!");
        exit(1);
      }
      break;
    case 0x0103:          /* Compression     */
      if (value != 1)
      {
        printf("Compression Used! Error!\n");
        exit(1);
      }
      break;
    case 0x0106:          /* Black is        */
      black = value;
      break;
    case 0x0107:          /* Thresholding tag */
      if ((value < 1) || (value > 3))
      {
        printf("Thresholding Tag = Illegal value!\n");
        exit(1);
      }
      break;
    case 0x0111:          /* Offset to Raster Data */
      raster_offset = value;
      break;
    case 0x0115:
      bps = value;
      break;
    case 0x011a:
      break;
    case 0x011b:
      break;
    case 0x0128:
      break;
    default:
      if ((value & 32768) == 0) {
        printf("0x%04x Warning: An unsupported tag appears in the TIF file\n");
      }
      break;
  }
}

while(tagfile_position < raster_offset)
{
  i = getbyte(tagfile, &tagfile_position);
}

/*.....  READ IN FILE  .....*/
all_length = (int) ((length*width*bps)/(8*COMPRESSION_RATIO));
all_width = (int)((width*bps)/(8*COMPRESSION_RATIO));
/* printf("all_length = %d, all_width = %d \n", all_length, all_width); */

page = image_allocate((int) all_length, (int) all_width);

/*
printf("page-->rast = %d page-->size = %d \n", page->rast, page->size);
printf("Block = %d \n", block);
*/
for (i = 0; i<length; i++)
{
  for (j = 0; j<width; j++)
  {
    if (j%8) == 0)
    {
      v = getc(tagfile);
      tagfile_position++;
```

【図54】

```
        if (v == EOF)
        {
          printf("\n Error: Premature End of File found in tiff file\n");
          exit(1);
        }
      }
      if ((v & 128) == black*128)
        page->pixel[i][j] = 0;
      else
        page->pixel[i][j] = 1;
      v = v << 1;
    }
  }

  fclose(tmpfile);
  return page;
}
```

【図217】

Source Code for

Mono-Spaced ("Courrier") Segmentation

【図86】

```
            release_block(tmp1);
            tmp1 = tmp11;
        }
        for (; tmp2 != NULL; ){
            tmp11 = tmp2->next;
            release_block(tmp2);
            tmp2 = tmp11;
        }
        free_rectblock(initial_block);
        return;
}
/*.....*/
/*******  Draw text components for display                    *******/
/*.....*/

void block_circumscribe(p, width, height, p_ori)
unsigned char **p;
int          width, height;
unsigned char **p_ori;
{

        int i, j;
        /* open file for storing block data */

        original_p = p_ori;
        reduce_factor = SCALE_RATIO;

        for (i = 0; i <= 8; i++)
            for (j = 0; j <= 8; j++)
                direct_static[i][j] = 0;

        block_file = fopen("block_file.dat", "w");


        text_length = 150.0/SCALE_RATIO;
        total_text = 0;

        /* the root block stands for the whole page image unit */

        root_rectblock = new_block(0,0);
        root_rectblock->width = width;
        root_rectblock->length = height;
        root_rectblock->upper_y = 0;
        root_rectblock->lower_y = height-1;
        root_rectblock->left_x = 0;
        root_rectblock->right_x = width-1;

        /* find blocks inside the page range */
        block_in_range(p, 0, 0,  width-1, height-1, root_rectblock);
printf("get out of block_in_range\n");
        rearrange_rectblock(root_rectblock);
printf("get out of rearrange_rectblock\n");

        text_discriminate(root_rectblock);

printf("get out of text_dis\n");

        firstlevelnontextsearch(p, root_rectblock);

printf("get out of firstlevelnontextsearch\n");
        print_block(1, root_rectblock);


        fclose(block_file);
printf("before get out of block_construct\n");
```

```
/*      release_block(root_rectblock);*/
}
```

【図55】

```
/*****************************************************************************/
/*                                                                           */
/*      Filer block_main.c                                                   */
/*      Author  Shinichi Mung                                                */
/*      Date    3-Jul-1993                                                   */
/*      Copyright (c) 1993 Canon Information Systems                         */
/*                                                                           */
/*****************************************************************************/

#include "block_main.h"
#include "blockgeneral.h"
#include "image.h"

extern struct block_rect *root_rectblock;
struct linkblock *firstlinkblock;
struct sectionblock *firstsectionblock;

extern unsigned char **centrix();
extern struct block_image *readtiff();
extern void block_circumscribe();
extern void inline_block();
extern void in_section_block();
extern struct linkblock *sec_linkblock();
extern struct rect_block *sec_rectblock();
extern void print_linkblock();
extern void print_rectblock();
extern void free_pair();
extern void free_rectblock();

int     skew = 0;
struct block_image reduce_image;
struct image *page;
float scale_displayform;
unsigned char **b_image;

/* unsigned char **block_matrix; */

/*****************************************************************************/
/*                                                                           */
/*      Binarize the image                                                   */
/*                                                                           */
/*****************************************************************************/

void get_rid_of_speppers(pixel, first_x, second_x, second_y)
unsigned char **pixel;
int     first_x, second_x, second_y;
{
    int i, j;

    for (i = first_x; i <= second_x ; i++)
        for (j = first_y; j < second_y; j++){
            if ((pixel[i][j] == 1)&&
                (pixel[i][j-1]==0)&&
                (pixel[i-1][j-1]==0)&&
                (pixel[i-1][j]==0)&&
                (pixel[i-1][j+1]==0)&&
                (pixel[i][j+1]==0)&&
                (pixel[i+1][j+1]==0)&&
                (pixel[i+1][j]==0)&&
                (pixel[i+1][j-1]==0))
                pixel[i][j] = 0;
        }

/*****************************************************************************/
/*                                                                           */
/*      reduce the image                                                     */
/*                                                                           */
/*****************************************************************************/

void create_reduce_image()
{   int i, j, r;
    int x, y;

    reduce_image.width = page->size_x/SCALE_RATIO+2;
    reduce_image.height = page->size_y/SCALE_RATIO+2;

    reduce_image.pixel = centrix(0, reduce_image.height-1, 0, reduce_image.width-1);

    /* method 4 */

    for ( i = 0 ; i < reduce_image.height - 2; i++)
        for ( j = 0; j < reduce_image.width - 2 ; j++)
            y = i*SCALE_RATIO;
            x = j*SCALE_RATIO;
            if ((page->pixel[y+yoff[r][0]][x+xoff[r][0]]&0x01)&&
                (page->pixel[y+yoff[r][1]][x+xoff[r][1]]&0x01))
                {
                    reduce_image.pixel[i+1][j+1] = 1;
                    break;
                }
        }

    get_rid_of_speppers(reduce_image.pixel, 1,  1,
                        reduce_image.width-2,  reduce_image.height-2);
}

/*****************************************************************************/
/*                                                                           */
/*      Display the specified image matrix into the window                   */
/*                                                                           */
/*****************************************************************************/

void display_matrix(p, xmin, ymin, xmax, ymax, width, height, wd, color)
unsigned char **p;
int     xmin, ymin, xmax, ymax, width, height, color;
Widget  wd;
{
    float scale;
    int   x, y, x1, y1;

    /*
    printf("get into display \n");

    printf("width = %d height = %d xmin = %d ymin = %d xmax = %d ymax = %d\n",
            width, height, xmin, ymin, xmax, ymax);
    */
    scale_displayform = (float)(ymax-ymin+1)/(float)(xmax-xmin+1);
    scale = (float)height/(float)width;

/*  printf("scale_displayform = %f scale = %f \n", scale_displayform, scale); */
    if (scale_displayform > scale)
        width = (int)((float)height/(float)scale_displayform);
    else
        height = (int)((float)width*scale_displayform);

    scale_displayform = (float)(ymax-ymin+1)/(float)height;
    if (scale_displayform < 1.0){
        scale_displayform = 1.0;
        width = (xmax-xmin+1);
```

【図56】

```
    height = (ymax-ymin+1);
  }
/*
printf("width = %d height = %d scale_displayform = %f\n",
    width, height, scale_displayform);
*/
  if (color == 0){
    for ( y = 0; y < height; y++){
      printf("y = %d \n", y);
      for ( x = 0; x < width; x++){
        x1 = (int)((float)x*scale_displayform)+xmin;
        y1 = (int)((float)y*scale_displayform)+ymin;
        if (pix[y][x][0]>0)
          xs_color_point(wd, x+1, y+1, color);
      }
    }
    printf("after plot \n");
  } else {
    for ( y = 0; y < height; y++){
      for ( x = 0; x < width; x++){
        x1 = (int)((float)x*scale_displayform)+xmin;
        y1 = (int)((float)y*scale_displayform)+ymin;
        if (pix[y][x][0]>0)
          xs_color_point(wd, x+1, y+1, color);
      }
    }
    printf("after flush \n");
  }
  xs_flush(wd);
  printf("after flush \n");
}

/*****************************************************************/
/*                                                               */
/*          Display the rectblock                                */
/*                                                               */
/*****************************************************************/

void display_rectblock(firstblock, wd)
struct block_rect *firstblock;
Widget wd;
{
  struct block_rect *tmpb;
  int  x1, y1, x2, y2;
/*
printf("scale_display =%f\n", scale_displayform);
*/
  if (firstblock->rect[FAMILY])
    xs_set_foreground(FAMILYTEXT_COLOR, wd);
  else
    xs_set_foreground(TEXT_COLOR, wd);
  for (tmpb = firstblock;tmpb != NULL; tmpb = tmpb->next){
    x1 = (int)((float)tmpb->rect[0]*scale_displayform);
    y1 = (int)((float)tmpb->rect[1]*scale_displayform);
    x2 = (int)((float)tmpb->rect[2]*scale_displayform);
    y2 = (int)((float)tmpb->rect[3]*scale_displayform);
    if (x1 < 1) x1 = 1;
    if (y1 < 1) y1 = 1;
    if (x2 < 1) x2 = 1;
    if (y2 < 1) y2 = 1;
    xs_draw_box(wd, x1, y1, x2-1, y2-1);
  }

  for (tmpb = firstblock->firstnonstrblock; tmpb != NULL; tmpb = tmpb->next){
    x1 = (int)((float)tmpb->rect[0]*scale_displayform);
    y1 = (int)((float)tmpb->rect[1]*scale_displayform);
    x2 = (int)((float)tmpb->rect[2]*scale_displayform);
    y2 = (int)((float)tmpb->rect[3]*scale_displayform);
    if (x1 < 1) x1 = 1;
    if (y1 < 1) y1 = 1;
    if (x2 < 1) x2 = 1;
    if (y2 < 1) y2 = 1;
    xs_draw_box(wd, x1, y1, x2-1, y2-1);
  }

  if (firstblock->xfirstnonstrblock) {
    x1 = (int)((float)tmpb->rect[0]*scale_displayform);
    y1 = (int)((float)tmpb->rect[1]*scale_displayform);
    x2 = (int)((float)tmpb->rect[2]*scale_displayform);
    y2 = (int)((float)tmpb->rect[3]*scale_displayform);
    xs_draw_box(wd, x1, y1, x2-1, y2-1);
  }
}

/*****************************************************************/
/*                                                               */
/*          Display CAP LINE                                     */
/*                                                               */
/*****************************************************************/

void display_capline(wd)
Widget wd;
{
  int x1, y1, x2, y2;

  xs_set_foreground(LINECAP_COLOR, wd);

  for (x = 0; x < reduce_image_width; x++)
    for (y = 0; y < reduce_image_height; y++) {
      if (reduce_image_pixel[y][x][0]>0) {
        x1 = (int)((float)x*scale_displayform);
        y1 = (int)((float)y*scale_displayform);
        xs_plot(wd, x1+1, y1+1);
      }
    }
}

/*****************************************************************/
/*                                                               */
/*          Display the lineblock                                */
/*                                                               */
/*****************************************************************/

void display_lineblock(firstline, wd)
struct lineblock *firstline;
Widget wd;
{

  struct lineblock *tmpl;
  int   x1, y1, x2, y2;
/*
printf("get into lineblo \n");
*/
  if (firstline->xtf[FAMILY])
    xs_set_foreground(FAMILYTEXT_COLOR, wd);
  else
    xs_set_foreground(TEXT_COLOR, wd);
```

【図57】

```
for (tmpl = firstline->firstsubline; tmpl != NULL; tmpl = tmpl->next){
    x1 = (int)((float)tmpl->left_x/scale_displayform);
    y1 = (int)((float)tmpl->upper_y/scale_displayform);
    x2 = (int)((float)tmpl->width/scale_displayform);
    y2 = (int)((float)tmpl->length/scale_displayform);
    if (x1 < 1) x1 = 1;
    if (y1 < 1) y1 = 1;
    if (x2 < 1) x2 = 1;
    if (y2 < 1) y2 = 1;
    xs_draw_box(wd, x1, y1, x1+x2, y1+y2);
    if (tmpl->attrCAPTION) xs_draw_line(wd, x1, y1, x1+5, y1+5);
}

for (tmpl = firstline->firstsubline; tmpl != NULL; tmpl = tmpl->next){
    x1 = (int)((float)tmpl->left_x/scale_displayform);
    y1 = (int)((float)tmpl->upper_y/scale_displayform);
    x2 = (int)((float)tmpl->width/scale_displayform);
    y2 = (int)((float)tmpl->length/scale_displayform);
    if (x1 < 1) x1 = 1;
    if (y1 < 1) y1 = 1;
    if (x2 < 1) x2 = 1;
    if (y2 < 1) y2 = 1;
    if (tmpl->attrFRAME)
        xs_set_foreground(FRAME_COLOR, wd);
    else if (tmpl->attrPICTURE)
        xs_set_foreground(PICTURE_COLOR, wd);
    else if (tmpl->attrLINE)
        xs_set_foreground(LINE_COLOR, wd);
    else if (tmpl->attrTABLE)
        xs_set_foreground(TABLE_COLOR, wd);
    else if (tmpl->attrWORLD||tmpl->attrLOCAL)
        xs_set_foreground(WORLD_LOCAL_COLOR, wd);
    else if (tmpl->attrPICTURE)
        xs_set_foreground(PICTURE_COLOR, wd);
    else
        xs_set_foreground(UNKNOWN_COLOR, wd);
    xs_draw_box(wd, x1, y1, x1+x2, y1+y2);
    if (tmpl->attrLOCAL||tmpl->attrWORLD||tmpl->attrPICTURE){
        xs_draw_box(wd, x1+3, y1+3, x1+x2-3, y1+y2-3);
        if (!(tmpl->attrWORLD||tmpl->attrPICTURE||
              (firstline->attrTABLE&&
               (firstline->attrFRAME||firstline->attrLINE)))
            continue;
    }
    else
        display_lineblock(tmpl, wd);
}

/*****************************************************************/
/*********  draw one side of box                         *********/
/*****************************************************************/
void draw_side(section, wd, color, side)
struct sectionblock *section;
Widget wd;
int    color, side;
{
    int x1, x2, y1, y2;

    x1 = (int)((float)section->left_x/scale_displayform);
    x2 = (int)((float)section->right_x/scale_displayform);
    y1 = (int)((float)section->upper_y/scale_displayform);
    y2 = (int)((float)section->lower_y/scale_displayform);

    xs_set_foreground(color, wd);
    switch(side){
        case 1:  xs_draw_line(wd, x1, y1-2, x2, y1-2);
                 xs_draw_line(wd, x1, y1-1, x2, y1-1);
                 break;
        case 2:  xs_draw_line(wd, x2-2, y1, x2-2, y2);
                 xs_draw_line(wd, x2-1, y1, x2-1, y2);
                 break;
        case 3:  xs_draw_line(wd, x1, y2+2, x2, y2+2);
                 xs_draw_line(wd, x1, y2+1, x2, y2+1);
                 break;
        case 4:  xs_draw_line(wd, x1-2, y1, x1-2, y2);
                 xs_draw_line(wd, x1-1, y1, x1-1, y2);
                 break;
    }
}

/*****************************************************************/
/*********  display the lineblock                        *********/
/*****************************************************************/
void display_lineblock(firstsection, wd)
struct sectionblock *firstsection;
Widget wd;
{

    struct sectionblock *tmps;
    int   x1, y1, x2, y2, i1, i2;

    if (firstsection->attrTABLE)
        xs_set_foreground(TABLEFRAME_COLOR, wd);
    else
        xs_set_foreground(TEXT_COLOR, wd);
    for (tmps = firstsection->firstsubsection; tmps != NULL; tmps = tmps->next){
        x1 = (int)((float)tmps->left_x/scale_displayform);
        y1 = (int)((float)tmps->upper_y/scale_displayform);
        x2 = (int)((float)tmps->width/scale_displayform);
        y2 = (int)((float)tmps->length/scale_displayform);
        if (x2 < 1) x2 = 1;
        if (y2 < 1) y2 = 1;
        xs_draw_box(wd, x1, y1, x1+x2, y1+y2);
        if (tmps->attrTITLE){
            if (y2 > 3){
                xs_draw_box(wd, x1+1, y1+1, x1+x2-1, y1+3);
            }
        }
        if (tmps->attrCAPTION)
            xs_draw_line(wd, x1, y1, x1+5, y1+5);
    }
}

for (tmps=firstsection->firstsubsection; tmps != NULL; tmps = tmps->next){
    x1 = (int)((float)tmps->left_x/scale_displayform);
    y1 = (int)((float)tmps->upper_y/scale_displayform);
    x2 = (int)((float)tmps->width/scale_displayform);
    y2 = (int)((float)tmps->length/scale_displayform);
    if (x2 < 1) x2 = 1;
    if (y2 < 1) y2 = 1;

    if (tmps->attrTERM)
        xs_set_foreground(TERM_COLOR, wd);
    else if (tmps->attrFRAME)
        xs_set_foreground(FRAME_COLOR, wd);
    else if (tmps->attrPICTURE)
        xs_set_foreground(PICTURE_COLOR, wd);
    else if (tmps->attrLINE)
        xs_set_foreground(LINE_COLOR, wd);
    else if (tmps->attrTABLE)
        xs_set_foreground(TABLEFRAME_COLOR, wd);
    else if (tmps->attrWORLD||tmps->attrLOCAL)
        xs_set_foreground(WORLD_LOCAL_COLOR, wd);
    else if (tmps->attrPICTURE)
        xs_set_foreground(PICTURE_COLOR, wd);
    else if (tmps->attrUNKNOWN)
        xs_set_foreground(UNKNOWN_COLOR, wd);
```

【図58】

```
xs_draw_box(wd, x1, y1, w1-1, l1-1);
if ((tmpa->attr&TEXT)!=attr){
  x1 = (int)((float)tmpa->left_x/scale_display(scm));
  y1 = (int)((float)tmpa->upper_y/scale_display(scm));
  x2 = (int)((float)tmpa->right_x/scale_display(scm));
  y2 = (int)((float)tmpa->lower_y/scale_display(scm));
  xs_draw_line(wd, x1, y1, x2, y2);
  xs_string(wd, x1, y1, "p");
}
else if ((tmpa->attr&TEXT)!=(TEXT|MASTER)){
  x2 = (int)((float)tmpa->right_x/scale_display(scm));
  y2 = (int)((float)tmpa->lower_y/scale_display(scm));
  xs_draw_line(wd, x1, y1, x2, y2);
  xs_string(wd, x1, y1, "p");
}
else if ((tmpa->attr&TEXT)&&(wd >= 3)&&( 12 >= 3)){
    xs_draw_box(wd, x1-1, y1-1, w1-2, l1-2);
if ((tmpa->attr&(MOVOID|TEXT))==(MOVOID|TEXT)){
  if (tmpa->attr&M)
    draw_side(tmpa, wd, LINE_COLOR, 1);
  else if (tmpa->attr&DOWN)
    draw_side(tmpa, wd, LINE_COLOR, 2);
  else if ((tmpa->attr&M)==(MOVOID|TEXT)){
    if (tmpa->attr&M)
      draw_side(tmpa, wd, LINE_COLOR, 2);
      else if (tmpa->attr&DOWN)
        draw_side(tmpa, wd, LINE_COLOR, 4);
    }
  }
  if (tmpa->attr&(EXPICTURE|VOCOR|MOVOID|PICTURE))
    continue;
  else
    display_sectionblock(tmpa, wd);
  }
}
/*-------------------------------------------------------------*/
/*-------- free firstblock ------------------------------------*/
/*-------------------------------------------------------------*/
void free_block(firstblock)
struct block_rect *firstblock;
{
  struct block_rect *tmp2;
  struct voiceblock *tmpa1, *tmpa2;
  int   i;
/*
printf("pac int free rect = %d x - %d y - %d \n", firstblock->index,
        firstblock->left_x, firstblock->upper_y);
*/
  for (tmpa1 = firstblock->firstsubblock; tmpa1 != NULL; ){
/* printf("tmp 1 = %d \n", tmpa1); */
  tmpa2 = tmpa1->next;
    free_voiceblock(tmpa1);
/* printf("next tmp2 = %d\n", tmpa2); */
  tmpa1 = tmpa2;
  }
  for ( tmp2 = firstblock->firstconnectblock; tmp2 != NULL; ){
    tmpa2 = tmp2->next;
    free_block(tmp2);
/* printf("free next = %d \n", tmp2); */
    tmp2 = tmpa2;
  }

  if (firstblock->firstwire != NULL){
/* printf("firstwire = %d \n", firstblock->firstwire); */

  for (tmpa1 = firstblock->firstwire; tmpa1 != NULL; ){
/* printf("free wire = %d \n", tmpa1->number); */
    tmpa2 = tmpa1->next;
    free_wire(tmpa1);
  tmpa1 = tmpa2;
  }
  free_rectblock(firstblock);
}

/*-------------------------------------------------------------*/
/*-------- free section ---------------------------------------*/
/*-------------------------------------------------------------*/
void free_section(firstsection)
struct sectionblock *firstsection;
{
  struct sectionblock *tmpa1, *tmpa2;
  struct linablock    *tmp1;
  struct block_rect   *tmpr1, *tmpr2;
/*
printf("get into free section \n");
*/
  for (tmpa1 = firstsection->firstsubsection; tmpa1 != NULL; ){
    tmpa2 = tmpa1->next;
/*
printf("section = %d \n", tmpa1, tmpa1->index);
*/
    for (tmp1 = tmpa1->firstlinablock; tmp1 != NULL; ){
      tmp2 = tmp1->next;
/*
printf("tmp1 = %d line = %d first = %d current %d \n", tmp1,
        tmp1->index, tmp1->first_block->index,
          tmp1->current_block->index);
*/
      for (tmpr1 = tmp1->first_block; tmpr1 != NULL; ){
        tmpr2 = tmpr1->next;
        free_block(tmpr2);
/*
printf("after free rect. tmpr2 = %d tmp1 = %d \n", tmpr2, tmp1);
*/
        tmpr1 = NULL;
        tmpr1 = tmpr2;
      }
      tmp1->first_block = NULL;
/*
printf("free line tmp1 = %d tmp1->index = %d undersd r_cap=%d\n",
    tmp1, tmp1->index, tmp1->under_caption, tmp1->r_caption);
*/
      free((struct linablock *)tmp1);
/*
printf("after free line tmp1 = %d \n", tmp1);
*/
```

【図59】

```
        tmp1 = NULL;
        tmp2 = tmp1;
    }
/*
printf("free section tmp1 = %d \n", tmp1);
*/
    free((struct sectionblock *)tmp1);
    tmp1 = NULL;
    tmp1 = tmp2;
  }
  for (tmp1 = firstsection->firstcontentsection; tmp1 != NULL; ){
/*
printf("content section = %d \n", tmp1->index);
*/
    tmp2 = tmp1->next;
    for (tmp3 = tmp1->firstlineblock; tmp3 != NULL; ){
      tmp4 = tmp3->next;
      free((struct linblock *)tmp3);
      tmp3 = tmp4;
    }
    free((struct sectionblock *)tmp1);
    tmp1 = tmp2;
  }
/*
printf("free line \n");
*/
  tmp1 = tmp2;
  for (tmp1 = tmp1->firstlineblock; tmp1 != NULL; ){
    tmp2 = tmp1->next;
    free((struct linblock *)tmp1);
    tmp1 = tmp2;
  }
/*
printf("free section \n");
*/
  free((struct sectionblock *)firstsection);
}

/*****************************************************************/
/********** Open tiff file and reduce its size by 2 **************/
/*****************************************************************/
void open_tiff_file (filename)
char filename[];
{

  printf("step = %d \n", step);
    if (step != 0){
      free_matrix(reduce_image.pixel, 0, reduce_image.height-1,
                                          0, reduce_image.width-1);
      printf("before  free section \n");
      if (step==2_CONN){
        free_matrix(h_image, 0, reduce_image.height-1,
                                          0, reduce_image.width-1);
      }
      free_section((firstsectionblock));
      printf("after free section \n");
    }
    image_deallocate(page);
    step = 0;
  }
  printf("\nreading....");
  page = readtiff(filename);
  printf("done. \n");
  get_tiff_paper(page->pixel, 1.2, page->xisc-2, page->ysc-2);
/*
printf("line = %d first = %d current = %d \n", tmp1->index, tmp1->first_block->ydistf("xisc = %d ysc = %d \n", page->xisc, page->ysc);
*/
    create_reduce_image();
    step = 0_XISC|0_YSC;
}

/*****************************************************************/
/********** get text block into text display *********************/
/*****************************************************************/

void get_text(tmpsect, p, wd, color)
struct sectionblock *tmpsect;
unsigned char **p;
Widget wd;
int   color;
{
    struct linblock *tmp1;
    struct block_rect *tmp2;
    int x, y, x1, y1;

    xs_set_foreground(color, wd);
    for (tmp1 = tmpsect->firstlineblock; tmp1 != NULL; tmp1 = tmp1->next){
      for (tmp2 = tmp1->first_block; tmp2 != NULL; tmp2 = tmp2->next){
        for (y = tmp2->ylist; y <= tmp2->lower; y++){
          for (x = tmp2->xlist; x <= tmp2->right; x++){
            if (reduce_image.pixel[y][x] == text_block){
              plot(x, y);
              x1 = (int)((float)x/scale_display(form));
              y1 = (int)((float)y/scale_display(form));
              xs_plot(wd, x1, y1);
            }
          }
        }
      }
    }
}

/*****************************************************************/
/********** get text into table text display *********************/
/*****************************************************************/

void get_thistext(tmpsect, p, wd)
struct sectionblock *tmpsect;
unsigned char **p;
Widget wd;
{
    int   x, y, x1, y1;
```

【図60】

【図61】

```
    }
  }
  else if (tmpsel->attb[VERTICALSOLID]){
      get_line(tmpsel, b_image, LINE_BLOCK, LINE_COLOR, wd);
      get_line(tmpsel, b_image, LINE_BLOCK, LINE_COLOR, wd);
    else if (tmpsel->attb[PICTURE]){
        get_picture(tmpsel, b_image,
                    PICTURE_BLOCK, PICTURE_COLOR, wd);
    }
    else if (tmpsel->attb[TABLE]{ /*firstsubsection*/
        for (tmpsel2 = tmpsel->firstsubsection;
             tmpsel2 != NULL;
             tmpsel2 = tmpsel2->next)
          get_table(tmpsel2, b_image, wd);
        tmpsel->firstsubsection = NULL;
        separate_image(tmpsel, wd);
        tmpsel->firstsubsection = tmpsel2;
        get_picture(tmpsel, b_image, TABLE_BLOCK,
                    TABLE_COLOR, wd);
    } else if (tmpsel->attb[FRAME]){
        separate_image(tmpsel, wd);
        get_picture(tmpsel, b_image, FRAME_BLOCK,
                    FRAME_COLOR, wd);
    }
    else if (tmpsel->attb[PICTURE]{ b_image, wd);
        get_line(tmpsel->firstsubsection,
        for (tmpsel = tmpsel->firstsubsection;
             tmpsel != NULL; tmpsel = tmpsel->next)
          get_picture(tmpsel, b_image, wd, TEXT_COLOR);
    } else if (tmpsel->attb[UNKNOWN]
        get_picture(tmpsel, b_image,
                    UNKNOWN_BLOCK, UNKNOWN_COLOR, wd);
}

/*.....*/
/*..... block processing .....*/
/*.....*/

void block_process(wd, width, height)
Widget wd;
int  width, height;
{
    int yes;

/*
printf("get into process \n");
*/
    init_graphics(wd);
    xs_clr_win(wd);

    display_matrix(reduce_image.pixel, 0, 0, reduce_image.width-1,
                   reduce_image.height-1, width-2, height-2, wd, 0);

    if ((tmpsel->attb[CONN])){
      for (yes = 0 ; yes == 0; ) {
          printf("go for connected block generation? (1: yes, 0: no) ");
          fscanf(stdin, "%d", &yes);
      }

    printf("step 1 .....\n");
    block_firstsmallside(reduce_image.pixel, reduce_image.width,
                         reduce_image.height, page->plist);

    xs_clr_win(wd);
    display_textblock(root_textblock, wd);
    xs_flush(wd);

    for (yes = 0 ; yes == 0; ) {
        printf("go for line block generation? (1: yes, 0: no) ");
        fscanf(stdin, "%d", &yes);
    }

    printf("step 2 .....\n");
    firstlineblock = new_lnblock(0.0);

    ini_line_block(firstlineblock, root_textblock, reduce_image.width,
                   page->plist, reduce_image.pixel);

    print_lineblock(firstlineblock);

    xs_clr_win(wd);
    display_lineblocks(firstlineblock, wd);

    display_spline(wd);

    xs_flush(wd);

    for (yes = 0 ; yes == 0; ) {
        printf("go for section block generation? (1: yes, 0: no) ");
        fscanf(stdin, "%d", &yes);
    }

    printf("step 3 .....\n");
    firstsectionblock = new_secblock(0,0);
    ini_section_block(firstsectionblock,firstlineblock, reduce_image.pixel,
                      reduce_image.width);

    print_secblock(firstsectionblock);

    step 1- SP_CONN;
  }

  xs_clr_win(wd);
  display_sectionblock(firstsectionblock, wd);
  xs_flush(wd);

  for (yes = 0 ; yes == 0; ) {
      printf("go for color image generation? (1: yes, 0: no) ");
      fscanf(stdin, "%d", &yes);
  }

  xs_clr_win(wd);
  b_image = matrix(0, reduce_image.height-1, 0, reduce_image.width-1);

  separate_image(firstsectionblock, wd);
  xs_flush(wd);
}

/*.....*/
/*..... color display .....*/
/*.....*/

void color_reference(wd, width, height)
Widget wd;
```

[図63]

```
/*****************************************************************************/
/*****************************************************************************/
/********** File:  mamalloc.c                                 **********/
/********** Author: Shin-Iwan Wang                            **********/
/********** Date :  10-1-91                                   **********/
/********** Copyright 1992 Canon Information Systems          **********/
/*****************************************************************************/
/*****************************************************************************/
#include <malloc.h>
#include <stdio.h>

void nrerror(error_text)
char error_text[];
{
        void exit();

        fprintf(stderr, "%s\n", error_text);
        exit(1);
}


/* Allocate a float vector with range [nl..nh] */
float *vector(nl, nh)
int nl, nh;
{
        float *v;

        v = (float *) malloc((unsigned)(nh-nl+1)*sizeof(float));
        if (!v) nrerror("allocation failure in vector()");
        return v-nl;
}

/* Allocate a unsigned char with range [nl..nh] */
unsigned char *cvector(nl, nh)
int nl, nh;
{
        unsigned char *v;

        v = (unsigned char *) malloc((unsigned)(nh-nl+1)*sizeof(unsigned char));
        if (!v) nrerror("allocation failure in cvector()");
        return v-nl;
}



/* Allocate a int vector with range [nl..nh] */
int  *ivector(nl, nh)
int nl, nh;
{
        int *v;
        int i;

        v = (int *) malloc((unsigned)(nh-nl+1)*sizeof(int));
        if (!v) nrerror("allocation failure in ivector()");
        v = v-nl;
        for ( i = nl; i <=nh; i++)
          v[i] = 0;
        return v;

}

/* Allocate a double vector with range [nl..nh] */
double *dvector(nl, nh)
int nl, nh;
{
        double *v;

        v = (double *) malloc((unsigned)(nh-nl+1)*sizeof(double));
        if (!v) nrerror("allocation failure in dvector()");

        return v-nl;

/* Allocate a character matrix with range[nrl..nrh][ncl..nch] */
unsigned char **cmatrix(nrl, nrh, ncl, nch)
int nrl, nrh, ncl, nch;
{
        int i, j;
        unsigned char **m;

  m=(unsigned char **)malloc((unsigned)(nrh-nrl+1)*sizeof(unsigned char *));
  if (!m) nrerror("allocation failure 1 in cmatrix()");
        m -= nrl;

  for (i = nrl; i <= nrh; i++){
  m[i] = (unsigned char *) malloc((unsigned)(nch-ncl+1)*sizeof(unsigned char));
  if (!m[i]) nrerror("allocation failure 2 in cmatrix()");
  m[i] -= ncl;
   for (j = ncl; j <= nch; j++)
     m[i][j] = 0;
  }

  return m;
}

/* Allocate a float matrix with range [nrl..nrh][ncl..nch]. */
float **matrix(nrl, nrh, ncl, nch)
int nrl, nrh, ncl, nch;
{
        int i;
        float **m;

        m = (float **) malloc((unsigned)(nrh-nrl+1)*sizeof(float *));
        if (!m) nrerror("allocation failure 1 in matrix()");
        m -= nrl;

        for (i = nrl; i <= nrh; i++){
          m[i] = (float *) malloc((unsigned)(nch-ncl+1)*sizeof(float));
          if (!m[i]) nrerror("allocation failure 2 in matrix()");
          m[i] -= ncl;
        }
        return m;
}



/* Allocate an int matrix with range [nrl..nrh][ncl..nch]. */
int **imatrix(nrl, nrh, ncl, nch)
int nrl, nrh, ncl, nch;
{
        int i;
        int **m;

        m = (int **) malloc((unsigned)(nrh-nrl+1)*sizeof(int *));
        if (!m) nrerror("allocation failure 1 in matrix()");
        m -= nrl;

        for (i = nrl; i <= nrh; i++){
          m[i] = (int *) malloc((unsigned)(nch-ncl+1)*sizeof(int));
          if (!m[i]) nrerror("allocation failure 2 in matrix()");
          m[i] -= ncl;
        }
        return m;
```

【図64】

```
}

/* Allocate a double matrix with range [nrl..nrh][ncl..nch]. */
double **dmatrix(nrl, nrh, ncl, nch)
int nrl, nrh, ncl, nch;
{
    int i;
    double **m;

    m = (double **) malloc((unsigned) (nrh-nrl+1)*sizeof(double *));
    if (!m) nrerror("allocation failure 1 in matrix()");
    m -= nrl;

    for (i = nrl; i <= nrh; i++) {
        m[i] = (double *) malloc((unsigned) (nch-ncl+1)*sizeof(double));
        if (!m[i]) nrerror("allocation failure 2 in matrix()");
        m[i] -= ncl;
    }
    return m;
}

/* deallocate a float vector */
void free_vector(v, nl, nh)
float *v;
int nl, nh;
{
    free((float*)(v+nl));
}

/* deallocate an int vector */
void free_ivector(v, nl, nh)
int *v, nl, nh;
{
    free((int*)(v+nl));
}

/* deallocate a double vector */
void free_dvector(v, nl, nh)
double *v;
int nl, nh;
{
    free((double*)(v+nl));
}

/* deallocate a float matrix */
void free_matrix(m, nrl, nrh, ncl, nch)
float **m;
int nrl, nrh, ncl, nch;
{
    int i;

    for (i = nrh; i >= nrl; i--) free((float*) (m[i]+ncl));
    free((float*) (m+nrl));
}

/* deallocate an int matrix */
void free_imatrix(m, nrl, nrh, ncl, nch)
int **m;
int nrl, nrh, ncl, nch;
{
    int i;

    for (i = nrh; i >= nrl; i--) free((int*) (m[i]+ncl));
    free((int*) (m+nrl));
}

/* deallocate an char matrix */
void free_cmatrix(m, nrl, nrh, ncl, nch)
unsigned char **m;
int nrl, nrh, ncl, nch;
{
    int i;

    for (i = nrh; i >= nrl; i--) free((char*) (m[i]+ncl));
    free((char*) (m+nrl));
}

/* deallocate a double matrix */
void free_dmatrix(m, nrl, nrh, ncl, nch)
double **m;
int nrl, nrh, ncl, nch;
{
    int i;

    for (i = nrh; i >= nrl; i--) free((double*) (m[i]+ncl));
    free((double*) (m+nrl));
}
```

【図65】

```
/*****************************************************************/
/*                                                               */
/*                File: block_construct.c                        */
/*                Author: Shih-Yuan Wang                         */
/*                Date : 1-15-92                                 */
/*                                                               */
/*****************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
#include "blockgeneral.h"
#include "blockrect.h"
#include "mem.h"

extern    void xs_draw_box();
void      exit();
struct    block_rect *rect_freeblock;
int       total_rect;
float     rect_length;

FILE      *fopen(), *fp, *block_file;
char      *instant[4] = {"skip", "left", "right", "ignore"};

int       direct_stack[9][9];
int       reduce_factor;
unsigned char *original_ps
                                                        .........*/
/*.....  Malloc of two integer array                     .........*/
/*.....                                                  .........*/
int main(a, b)
int a, b;
{
    if (a >= b) return a;
    else return b;
}

/*****************************************************************/
/*                                                               */
/*********  avg and std of an integer array                      */
/*                                                               */
/*****************************************************************/

void avg_std(n, no, avg, std)
int *n, *no;
float *avg, *std;
{
    int i, total;
    long total_square;

    total = 0; total_square = 0;
    *avg = 0.0; *std = 0.0;
    for (i = 0; i < *no; i++){
        total += n[i];
        total_square += n[i]*n[i];
    }
    *avg = (float)(total)/(float)*no;
    *std = (float)((double)total_square/(double)*no-(*avg)*(*avg));
}

/*****************************************************************/
/*                                                               */
/*********  Allocate memory for block_rect                       */
/*                                                               */
/*****************************************************************/

struct block_rect *new_block(nl, nb)
int nl, nb;
{
    int i;
    struct block_rect *v, **v;

    v = (struct block_rect *) malloc (unsigned)(nb-nl+1)*
                                    sizeof(struct block_rect));
    if (!v){
        printf("allocation error: new_block \n");
        exit(1);
    }
    v = v-nl;
    for (i = nl; i <=nb; i++) {
        v[i].previous = v[i].next = NULL;
        v[i].currentsubblock = v[i].firstsubblock = NULL;
        v[i].firstmatchblock = v[i].currentmatchblock = NULL;
        v[i].firstwhite = v[i].currentwhite = NULL;
        v[i].nl = unused;
        v[i].index = i;
        v[i].numsubblock = 0;
        v[i].splitpt = 0;v[i].pairleft = NULL;
        v[i].if_sorted = 0;
        v[i].cover = 0;
        v[i].group = 0;
        v[i].in_group = 0;
        v[i].density = 0.0;
    }
    return v;
}

/*****************************************************************/
/*                                                               */
/*********  Chain the block                                      */
/*                                                               */
/*****************************************************************/

struct block_rect *new_getblock(headblock)
struct block_rect *headblock;
{
    struct block_rect *n;

    n = new_block(0,0);
    if (headblock->firstsubblock == NULL)
        headblock->firstsubblock = n;
    else{
        headblock->currentsubblock->next = n;
        n->previous = headblock->currentsubblock;
    }
    headblock->currentsubblock = n;
    headblock->numsubblock++;
    return n;
}

/*****************************************************************/
/*                                                               */
/*********  Allocate memory for whiteblock                       */
/*                                                               */
/*****************************************************************/

struct whiteblock *new_white(nl, nb)
int nl, nb;
{
    struct whiteblock *v;
    int i;

    v = (struct whiteblock *) malloc((unsigned)(nb-nl+1)*
                                    sizeof(struct whiteblock));
    if (!v){
        printf("allocation error: whiteblock nl = %d nb = %d \n", nl, nb);
```

【図66】

【図67】

```
/*****************************************************************************/

void add_x_right(x, y, ylist)
int          x, y;
struct data_outline *ylist;
{
        struct data_outline *tmp;

        tmp = new_h_outline(x);
        if (ylist[y].total_right == 0){
            ylist[y].current_right =  tmp;
            ylist[y].headright = tmp;
        } else {
                ylist[y].current_right->next = tmp;
                ylist[y].current_right = tmp;
              }
        ylist[y].total_right++;
}

/*****************************************************************************/
/********  Adding one element to data_outline on left side           *********/
/*****************************************************************************/

void add_x_left(x, y, ylist)
int          x, y;
struct data_outline *ylist;
{
        struct data_outline *tmp;

        tmp = new_h_outline(x);
        if (ylist[y].total_left == 0){
            ylist[y].current_left =  tmp;
            ylist[y].headleft = tmp;
        } else {
                ylist[y].current_left->next = tmp;
                ylist[y].current_left = tmp;
              }
        ylist[y].total_left++;
}

/*****************************************************************************/
/*******  Adding element to data_outline                             *********/
/*****************************************************************************/

void add_y_list(x, y, location, ylist)
int          x, y;
recording    location;
struct data_outline *ylist;
{
        if (location == LEFT)
           add_x_left(x, y,  ylist);
        else if (location == RIGHT)
           add_x_right(x, y, ylist);
             else if (location == BOTH){
                       add_x_left(x, y,  ylist);
                       add_x_right(x, y, ylist);
                  }
}

/*****************************************************************************/
/********  Sort integer array (from index nl to nh                   *********/
/*****************************************************************************/

void bubble_sort(list, nl, nh)
int *list;
int nl, nh;
{
    int i, k, tmp, stop_sort;

    stop_sort = 0;
    for ( i = 0; i < (nh-nl); i++){
          for(k = nl, stop_sort = 0; k < nh-i; k++)
             if (list[k] > list[k+1]){
                tmp = list[k];
                list[k] = list[k+1];
                list[k+1] = tmp;
                stop_sort = 1;
             }
          if (stop_sort == 0) break;
    }
}


/*****************************************************************************/
/********  Pair the element in two integer array;                    *********/
/*****************************************************************************/

void sort_x(pairright, pairleft)
int *pairright, *pairleft;
{
        int *temppair;
        int r,l,numpair;
        int i;

        numpair = (pairright[0] <= pairleft[0])?pairright[0]:pairleft[0];
/* printf("place 1 numpair = %d \n", numpair); */
        temppair = (int *)ivector(0, 2*numpair-1);
        bubble_sort(pairright, 1, pairright[0]);
        bubble_sort(pairleft, 1, pairleft[0]);

        for (numpair = 0, r = 1, l = 1; (r <= pairright[0])&&(l <= pairleft[0]); ){
            if ((l<pairleft[0])&&(pairleft[l] == pairleft[l-1])){
               l++;
               continue;
            }
            if ((r<pairright[0])&&(pairright[r] == pairright[r+1])){
               r++;
               continue;
            }
/*
            if ((pairright[r] == pairleft[l])||((pairright[r-1]==pairleft[l]))) {
               r++;
               l++;
               continue;
            }
*/
/*
            if (r < pairright[0]){
               if ((pairright[r]<pairleft[l])&&(pairright[r+1]>pairleft[l])){
                  temppair[numpair*2] = pairright[r];
                  temppair[numpair*2+1] = pairleft[l];
                  l++; r++; numpair++;
               } else if ((pairright[r]<pairleft[l])&&(pairright[r-1]<=pairleft[l])){
                         r++;
                      else if (pairright[r]>pairleft[l]) l++;
            } else {
                  if (pairright[r] < pairleft[l]){
                     temppair[numpair*2] = pairright[r];
                     temppair[numpair*2+1] = pairleft[l];
                     numpair++;
                     break;
                  } else l++;
               }
```

【図68】

```
    if (( pairright[0])){
        if (pairright[r] <= pairleft[1]){
            pairright[r] = pairright[r];
            tappair[numpair*2-1] = pairleft[1];
            for(r++ ; (r <= pairright[0])&&(pairright[r] <= pairleft[1]); r++);
            if (r <= pairright[0])
                for (l=1; (l<=pairleft[0])&&(pairright[r] > pairleft[l]); l++)
                    tappair[numpair*2+1] = pairleft[l];
            else
                tappair[numpair*2+1] = pairleft[pairleft[0]];
            numpair++;
        } else {
            if (pairright[r] <= pairleft[1]){
                tappair[numpair*2] = pairright[r];
                tappair[numpair*2+1] = pairleft[pairleft[0]];
                numpair++;
                break;
            } else r++;
        }
    }
    pairright[0] = pairleft[0] = numpair;
    for ( l = 1; l <= numpair; l++) {
        pairright[l] = tappair[l*2-1];
        pairleft[l] = tappair[l*2];
    }
    free((int *)tappair);
}

/*.....search the direction of next "one"                    .....*/
/*.....                                                      .....*/

void right_next_one(p, old_direct, l, r, x1, y1, x2, y2, new_direct, newx, newy,
                    location)
unsigned char **p;
int     old_direct;
int     l, r, x1, y1, x2, y2;
int     *new_direct, *newx, *newy;
recording
*location;
{
    int i, k;

    for ( i = 1; i <= search_map[old_direct][0] ; i++){
        *new_direct = search_map[old_direct][i];
        *newx = location_center[*new_direct].x;
        *newy = location_center[*new_direct].y;

        if ((p['newy]['newx] & 0x01)&&
            (*newx >= x1)&&(*newx <= x2)&&(*newy >= y1)&&(*newy <= y2)){
            *location = map_buff[old_direct][i];
            return;
        }
    }

    *newx = 0;
    *newy = 0;
    *location = SKIP;
}

/*.....                                                      .....*/
/*.....  Set up the extreme point for the matched white block .....*/
/*.....                                                      .....*/

void set_outline(x, y, nblock)
int     x, y;
struct nblock_set  *nblock;
{
    if (x > nblock->right_x ){
        nblock->right_x = x;
    } else if (x < nblock->left_x)
        nblock->left_x = x;

    if (y > nblock->lower_y)
        nblock->lower_y = y;
    else if (y < nblock->upper_y)
        nblock->upper_y = y;
}

/*.....                                                      .....*/
/*.....  Set up the extreme point for the matched block      .....*/
/*.....                                                      .....*/

void set_white_outline(x, y, nblock)
int     x, y;
struct block_set  *nblock;
{

    if (x > nblock->right_x )
        nblock->right_x = x;
    else if (x < nblock->left_x)
        nblock->left_x = x;

    if (y > nblock->lower_y)
        nblock->lower_y = y;
    else if (y < nblock->upper_y)
        nblock->upper_y = y;
}

/*.....                                                      .....*/
/*.....  Allocate int* memory                               .....*/
/*.....                                                      .....*/

int **intpointer(nl, nh)
int nl, nh;
{
    int **v;

    if((v = (int **) malloc((nh-nl+1)*sizeof(int *))) == NULL){
        printf("Allocation error in intpointer");
        exit(1);
    }

    v = v-nl;
    return v;
}

/*.....                                                      .....*/
/*.....  Transfer outline data into pair                     .....*/
/*.....                                                      .....*/

void put_outline_into_pair(nblock, ylist)
struct block_set    *nblock;
struct data_outline ylist[];
{
    int i, k;
    struct h_outline *tmp1, *tmp2;

    nblock->pairright = (int **)intpointer(nblock->upper_y, nblock->lower_y);
    nblock->pairleft  = (int **)intpointer(nblock->upper_y, nblock->lower_y);
    for (i = nblock->upper_y; i <= nblock->lower_y; i++){
```

【図69】

```
    nblock->pairright[i] = (int **)ivector(0, ylist[i].total_right);
    nblock->pairleft[i] = (int **)ivector(0, ylist[i].total_left);
    nblock->pairright[i][0] = ylist[i].total_right;
    for (tmp1 = ylist[i].headright, k = 1; tmp1 != NULL; k++){
        nblock->pairright[i][k] = tmp1->x;
        tmp1 = tmp1->next;
        free((struct h_outline *)tmp1);
        tmp1 = tmp;
    }
    nblock->pairleft[i][0] = ylist[i].total_left;
    for (tmp1 = ylist[i].headleft, k = 1; tmp1 != NULL; k++){
        nblock->pairleft[i][k] = tmp1->x;
        tmp = tmp1->next;
        free((struct h_outline *)tmp1);
        tmp1 = tmp;
    }
}

/*****************************************************************/
/*******  Transfer outline data into pair                  *******/
/*****************************************************************/

void put_whiteoutline_into_pair(nblock, ylist)
struct whiteblock  *nblock;
struct data_outline ylist[];
{
    int i, k;
    struct h_outline *tmp1, *tmp;

    nblock->pairright = (int **)intpointer(nblock->upper_y, nblock->lower_y);
    nblock->pairleft  = (int **)intpointer(nblock->upper_y, nblock->lower_y);

    for (i = nblock->lower_y; i <= nblock->upper_y; i++){
        nblock->pairright[i] = (int *)ivector(0, ylist[i].total_right);
        nblock->pairleft[i] = (int *)ivector(0, ylist[i].total_left);
        nblock->pairright[i][0] = ylist[i].total_right;
        for (tmp1 = ylist[i].headright, k = 1; tmp1 != NULL; k++){
            nblock->pairright[i][k] = tmp1->x;
            tmp = tmp1->next;
            free((struct h_outline *)tmp1);
            tmp1 = tmp;
        }
        nblock->pairleft[i][0] = ylist[i].total_left;
        for (tmp1 = ylist[i].headleft, k = 1; tmp1 != NULL; k++){
            nblock->pairleft[i][k] = tmp1->x;
            tmp = tmp1->next;
            free((struct h_outline *)tmp1);
            tmp1 = tmp;
        }
    }
}

/*****************************************************************/
/*******  Convert block into head block                    *******/
/*****************************************************************/

void convert_headblock(first)
struct block_rect *first;
{
    struct block_rect* m;

    if (first->firstsubblock != NULL) return;
    m = new_block(0,0);
    m->index = 1;
    m->upper_y = first->upper_y;
    m->lower_y = first->lower_y;
    m->left_x = first->left_x;
    m->right_x = first->right_x;
    m->x = first->x;
    m->y = first->y;
    m->width = first->width;
    m->length = first->length;
    m->no_point = first->no_point;
    m->act = first->act;
    m->density = first->density;
    m->if_rotated = first->if_rotated;
    m->pairright = first->pairright;
    m->pairleft = first->pairleft;
    first->pairright = NULL;
    first->pairleft = NULL;
    m->owner = ORIGINAL;

    first->nsubblock = 1;
    first->firstsubblock = m;
    first->currentsubblock = m;
    first->firstcontactblock = NULL;
/*printf("get out of convertblock\n");*/
}

/*****************************************************************/
/*******  Add one block under the specified block by the y location  *******/
/*****************************************************************/

void add_rectblock(tmp, first, head)
int tmp;
struct block_rect *first;
struct block_rect *head;
{
    struct block_rect* tmp2;

    if (first->upper_y < head->upper_y){
        head->upper_y = first->upper_y;
        head->length = head->lower_y-head->upper_y+1;
    }
    if (first->lower_y > head->lower_y){
        head->lower_y = first->lower_y;
        head->length = head->lower_y-head->upper_y+1;
    }
    if (first->left_x < head->left_x){
        head->left_x = first->left_x;
        head->width = head->right_x-head->left_x+1;
    }
    if (first->right_x > head->right_x){
        head->right_x = first->right_x;
        head->width = head->right_x-head->left_x+1;
    }
    if (tmp != TEST){
        if (head->firstsubblock == NULL){
            head->firstsubblock = first;
            first->previous = NULL;
            first->next = head->firstsubblock;
            head->currentsubblock = first;
        }
        else {
            for (tmp2 = head->currentsubblock; tmp2 != tmp->previous;){
                if (tmp2 == NULL) {
                    first->previous = NULL;
                    first->next = head->firstsubblock;
                    head->firstsubblock->previous = first;
```

【図70】

```
        head->firstcontextblock = first;
        break;
      }
      else if (first->upper_y < tmpb->upper_y) continue;
        else {
              first->previous = tmpb;
              first->next = tmpb->next;
              tmpb->next = first;
              if (first->next != NULL)
                  first->next->previous = first;
              else
                  head->currentcontextblock = first;
              break;
            }
      }
    } else {
        if (head->firstsubblock == NULL){
            head->firstsubblock = first;
            first->previous = first->next = NULL;
            head->currentsubblock = first;
        }
        else {
            for (tmpb = head->currentsubblock; ; tmpb = tmpb->previous){
                if (tmpb == NULL) {
                        first->previous = NULL;
                        first->next = head->firstsubblock;
                        head->firstsubblock->previous
                                    = first;
                        head->firstsubblock = first;
                        break;
                    }
                else if (first->upper_y < tmpb->upper_y) continue;
                    else {
                        first->previous = tmpb;
                        first->next = tmpb->next;
                        tmpb->next = first;
                        if (first->next != NULL)
                        first->next->previous = first;
                        else
                            head->currentsubblock = first;
                        break;
                    }
            }
        }
    }
}

/*****************************************************************/
/*                                                               */
/*      Remove this block                                        */
/*                                                               */
/*****************************************************************/

void remove_white(nblock, wblock)
struct block_rect *nblock;
struct whiteblock *wblock;
{
    if (nblock->firstwhite == wblock){
        nblock->firstwhite = wblock->next;
        if (wblock->next != NULL) wblock->next->previous = NULL;
        else nblock->currentwhite = NULL;
    }
    else if (nblock->currentwhite == wblock){
        nblock->currentwhite = wblock->previous;
        wblock->previous->next = NULL;
    } else {
        wblock->previous->next = wblock->next;
        wblock->next->previous = wblock->previous;
    }
}

/*****************************************************************/
/*                                                               */
/*      Remove text block                                        */
/*                                                               */
/*****************************************************************/

void remove_textblock(first, head)
struct block_rect *first, *head;
{
    if (head->firsttextblock == first){
        head->firsttextblock = first->next;
        if (first->next != NULL) first->next->previous = NULL;
        else head->currenttextblock = NULL;
    }
    else if (head->currenttextblock == first){
            head->currenttextblock = first->previous;
            first->previous->next = NULL;
        } else {
            first->previous->next = first->next;
            first->next->previous = first->previous;
        }
}

/*****************************************************************/
/*                                                               */
/*      Remove context block                                     */
/*                                                               */
/*****************************************************************/

void remove_contextblock(first, head)
struct block_rect *first, *head;
{
    if (head->firstcontextblock == first){
        head->firstcontextblock = first->next;
        if (first->next != NULL) first->next->previous = NULL;
        else head->currentcontextblock = NULL;
    } else if (head->currentcontextblock == first){
            head->currentcontextblock = first->previous;
            first->previous->next = NULL;
        } else {
            first->previous->next = first->next;
            first->next->previous = first->previous;
        }
}

/*****************************************************************/
/*                                                               */
/*      Transfer block from one head to another head block       */
/*                                                               */
/*****************************************************************/

void transfer_block(fromtext, totext, first, originalhead, head)
int fromtext, totext;
struct block_rect *first, *originalhead, *head;
{
/*printf("get into transfer blocknr"); */
    if (fromtext == TEXT)
        remove_textblock(first, originalhead);
    else
        remove_contextblock(first, originalhead);

add_newblock(totext, first, head);
if (originalhead != head) {
    originalhead->numblocks--;
```

【図71】

```
    head->nwwwbblock->;
  }
/*printf("get out of  transfer block\n");*/
}

/*.....*/
/*.....*/
/*.....*/

void rearrange_rectblock(headblock)
struct block_rect *headblock;
{
    struct block_rect *tmp, *tmp1, *tmpbb, *b;

/*  printf("get into rearrange_rectblock\n"); */

    tmp = headblock->firstpubblock;
    for  ( ; tmp != NULL; tmp = tmpbb){
      tmpbb = tmp->next;
      for ( tmp1 = tmp->next; tmp1 != NULL; tmp1 = tmp1->next){
        if (tmp->lower_y < tmp1->upper_y)
          break;
        else{
          if ((tmp->lower_y>=tmp1->lower_y)&&
              (tmp->left_x<=tmp1->left_x)&&
              (tmp->right_x >= tmp1->right_x)){
            tmp->cover = COVERED;
            tmp1->cover = COVER;
            transfer_block(TEXT, TEXT, tmp1, headblock, tmp);
            if (tmpbb == tmp1) tmpbb = tmp->next;
          }
          tmp1 = tmpbb;
        } else if ((tmp1->lower_y >= tmp->lower_y)&&
                   (tmp1->left_x <= tmp->left_x)&&
                   (tmp1->right_x >= tmp->right_x)&&
                   (tmp->upper_y >= tmp1->upper_y)){
                     convert_headblock(tmp1);
                     tmp1->cover = COVER;
                     if (tmp->cover == COVER){
                       for (b = tmp->firstpubblock; b != NULL;
                                                    b = b->next){
                         transfer_block(TEXT, TEXT, b, tmp, tmp1);
                         b->cover = COVERED;
                       }
                     }
                     remove_textblock(tmp, headblock);
                   } else {
                       tmp->cover = COVERED;
                       transfer_block(TEXT, TEXT, tmp,
                                      headblock, tmp1);
                       break;
                   }
      }
    }
/*  printf("get out of  rearrange_rectblock\n"); */
}

/*.....*/
/*.....  Search 8-connected "on" component                    .....*/
/*.....*/

void search_on(p, starty, x1, y1, x2, y2, nblock)

unsigned char     **p;
int               starty, x1, y1, x2, y2;
struct block_rect *nblock;
{
    int oldx, oldy, newx, newy;
    int olddirect, newdirect;
    recording location;
    struct data_outline *ylist_on;
    int i, j;

/* printf("get into search one x = %d y = %d\n", startx, starty); */

    ylist_on =(struct data_outline*)new_data_outline(y1, y2);

    nblock->x = startx;
    nblock->y = starty;
    nblock->left_x  = nblock->right_x = startx;
    nblock->upper_y = nblock->lower_y = starty;

    if (test_one_branch(p, startx, starty, DIST, x1, y1, x2, y2)==0){
      nblock->direction = nblock->length = 1; nblock->density = 1.0;
      p[starty][startx] = LEFTRIGHT_ONE;
      free((struct data_outline *)ylist_on->y);
      return;
    }
/*
    for ( i = starty-5; i <= starty+20; i++){
      printf("%3d ", i);
      for (j = startx-10; j <= startx+30; j++)
        printf("%c", p[i][j]);
      printf("\n");
    }
*/

    if (p[starty][startx-1]==0){
      add_y_list(startx, starty, RIGHT, ylist_on);
      p[starty][startx] = RIGHT_ONE;
    }
    else{
      add_y_list(startx, starty, LR, ylist_on);
      p[starty][startx] = LEFTRIGHT_ONE;
    }

    olddirect = SOUTH;
    right_post_pos(p, olddirect, oldx, oldy, x1, y1, x2, y2,
                   &newdirect, &newx, &newy, &location);
    set_putline(newx, newy, nblock);
    oldx = newx; oldy = newy;
    olddirect = newdirect;
    for ( ; ; ){
      do {
        right_post_pos(p, olddirect, oldx, oldy, x1, y1, x2, y2,
                       &newdirect, &newx, &newy, &location);

        if (location != SKIP)
          add_y_list(oldx, oldy, location, ylist_on);

        if ((location == SKIP)&&(p[oldy][oldx]==ONE))
          p[oldy][oldx] = LEFT;
        if (location == LEFT)
          p[oldy][oldx] = LEFT_ONE;

        if (location == RIGHT) {
          if (p[oldy][oldx] != LEFT_ONE)
            p[oldy][oldx] = RIGHT_ONE;
        }
```

【図72】

```
        if (location == RU) p[o][k][oldx] = LEFTSCORT_OVE;

        direct_class[oldx][newdirect]--;  */
/*      printf("%d %d %d %d\n", oldx, oldy, location[location]);*/
            set_outline(newx, newy, nblock);
            if (p[newy][newx] == OWN nblock->on_point++;
                oldx = newx;
                oldy = newy;
                olddirect = newdirect;
            } while ((newx != startx)||(newy != starty));
            if (test_one_branch(p, newx, newy, newdirect, x1, y1, x2, y2) == 0)
                break;
        }
        nblock->on_point++;

        put_outline_into_pair(nblock, ylist_on);
        check_around_right_object(ylist_on);
        nblock->length = nblock->lower_y - nblock->upper_y+1;
        return;
    }

/*****************************************************************/
/*******  Search the existence of surrounding 'zero'  ************/
/*****************************************************************/

int test_zero_branch(p, x, y, newdirect)
unsigned char **p;
int
    int x, y, newx, newy;
{
    x = search_xy[newdirect][search_xy[newdirect][0]];
    for (i = k+1; i <= EAST; i++){
        newx = x+around_center[i][0];
        newy = y+around_center[i][1];
        if (p[newy][newx] == ZERO){
            if ((i == EAST)||(i == SOUTH)) return 1;
            if (i == NEI{
                if ((p[newy-1][newx+1][0]!=4)||(p[newy][newx-1][0]!=1))
                    continue;
                else
                    return 1;
            }
            if (i== SEI{
                if ((p[newy-1][newx-1][0]!=4)||(p[newy][newx-1][0]!=1))
                    continue;
                else
                    return 1;
            }
            if (i == NWI{
                if ((p[newy][newx-1][0]!=4)||(p[newy+1][newx][0]!=1))
                    continue;
                else
                    return 1;
            }
            if (i == SWI{
                if ((p[newy+1][newx][0]!=4)||(p[newy][newx-1][0]!=1))
                    continue;
                else
                    return 1;
            }
        }
    }
    return 0;
}

/*****************************************************************/
/*******  Search the direction of next 'zero'  *******************/
/*****************************************************************/

void right_next_zero(p, old_direct, i, j, new_direct, newx, newy, location)
unsigned char **p;
int         old_direct;
int         i, j;
int         *new_direct, *newx, *newy;
recording   *location;
{
    int i;

    for ( i = i; i <= search_num[old_direct][0]; i--){
        *new_direct = search_num[old_direct][i];
        *newx = i+around_center[*new_direct].x;
        *newy = j+around_center[*new_direct].y;
        if ((p[*newy][*newx]&0x08) == ZERO){
            if ((*new_direct== NWI4
                ((p[*newy-1][*newx]&0x02)!=4&&(p[*newy][*newx-1]&0x01))
                continue;
            if ((*new_direct== NEI4
                ((p[*newy-1][*newx]&0x02)!=4&&(p[*newy][*newx+1]&0x01))
                continue;
            if ((*new_direct== WSI4
                ((p[*newy][*newx+1]&0x01)!=4&&(p[*newy+1][*newx]&0x02))
                continue;
            if ((*new_direct== ESI4
                ((p[*newy+1][*newx]&0x02)!=4&&(p[*newy][*newx-1]&0x01)))
                continue;
            *location = sup_inf[old_direct][i];
            return;
        }
    }
    *newx = -1;
    *newy = -1;
    *location = SKIP;
}

/*****************************************************************/
/*******  Search connected 'zero' component  *********************/
/*****************************************************************/

int search_zero(p, startx, starty, nblock)
unsigned char  **p;
int         startx, starty;
struct block_rect   *nblock;
{
    int         old, oldy, newx, newy;
    int         olddirect, newdirect, wide;
    recording   location;
    struct data_outline *ylist_off;
    struct whiteblock *emptywhite;

/* printf("enc_inc search zero x = %d y = %d \n", startx, starty); */

    wide = 8;
    if(test_zero_branch(p, startx, starty, EAST) == 0){
        printf("Island - RIGHTART_ZERO");
        return white;
    }
    ylist_off = (struct data_outline*)new_data_outline(nblock->upper_y,
                                                       nblock->lower_y);
    emptywhite = new_white(0,0);
    emptywhite->x = startx;
```

【図73】

```
    }

/*..... search block in white .....*/

int block_in_white(nblock, p, tapwhite)
struct block_rect *nblock;
unsigned char **p;
struct WhiteBlock *tapwhite;
{
    int i, j, k, gain, block_no, keep;
    struct block_rect *curr_block;

/*
printf("get into block_in_white\n");
*/
/*
    for ( i = tapwhite->upper_y; i <= tapwhite->lower_y; i++){
/*
printf("i = %d \n", i);
        for (j = 1; j <= tapwhite->pairright[i][0]; j++)
            printf("%d", tapwhite->pairright[i][j]);
        printf("\n");
*/
        for (j = 1; j <= tapwhite->pairleft[i][0]; j++)
            printf("%d", tapwhite->pairleft[i][j]);
        printf("\n");
*/
    curr_b=tapwhite->pairright[1];  tapwhite->pairleft[1];
/*
        printf("pair = %d \n", tapwhite->pairright[i][0]);
        for (j = 1; j <= tapwhite->pairright[i][0]; j++)
            printf("%d ", tapwhite->pairright[i][j]);
        printf("........");
        printf("\n");
        for (j = 1; j <= tapwhite->pairleft[i][0]; j++)
            printf("%d ", tapwhite->pairleft[i][j]);
        printf("\n");
    }
*/
    for (i = tapwhite->upper_y, block_no = 0; i <= tapwhite->lower_y; i++){
    for ( j = 1; j <= tapwhite->pairright[i][0]; j++){
/*
printf("i = %d left = %d \n", i, tapwhite->pairleft[i][j]);
printf("i = %d right = %d \n", i, tapwhite->pairright[i][j]);
*/
        for (k = tapwhite->pairleft[i][j]+1; gain = 0;
                                k < tapwhite->pairleft[i][j]+2; k++ ){
            if ((gain == 0) && (p[i][k] == RIGHT_ONE)) {
                gain = 1;
            } else if ((gain == 1)&&(p[i][k] == LEFT_ONE)){
                gain = 0;
            } else if ((gain == 0)&&(p[i][k] == ONE)){
                curr_block = new_setblock(nblock);
                search_omoji(p, k, i,tapwhite->left_x,
                             tapwhite->upper_y, tapwhite->right_x,
                             tapwhite->lower_y, curr_block);
/*
printf("after search_omoji\n");
*/
                curr_block->index = ++block_no;
                curr_block->in_group = nblock->group+1;
/*
printf("index = %d in_group = %d\n",curr_block->index, curr_block->in_group);
*/
                for ( i = (k-tapwhite->pairleft[i][j])+1;
                                    (p[i][k])&&(k<80); k++),
```

【図74】

```
                    k--;
                }
            }

        if (block_no == 0){
        for (i = capwhite->ypos_y; i <= capwhite->lower_y; i++)
            if (tap->pairinfo[i][0] > 1)
                tap->line_with_block++;
        }

        keep = 1;
        if ((block_no == 0)&&(tap->line_with_block <= 0.5*(POINT/SCALE_RATIO))
            keep = 0;
        /*
        printf("get out of block is valid keep = %d\n", keep);
        */
        return keep;
    }

/*****************************************************************************/
/*********  search block in block                                    *********/
/*****************************************************************************/

void search_block_in_block(nblock, p)
struct block_rect *nblock;
unsigned char **p;
{
    struct whiteblock *capwhite, *tap;
    int   i, block_no, keep;
    struct block_rect *tmpi;

    nblock->group = 0;
    for (tap=nblock->firstwhite; tap!=(w_NULL ); )
        {
        /*
        printf("capwhite index = %d subnub =%d\n", capwhite->index, capwhite->numsubwhite);
        */
            keep = 0;
            if (capwhite->numsubwhite != 0){
                for (i = 0; i < capwhite->numsubwhite; i++){
        /*
        printf("i = %d \n", i);
        */
                    if (block_in_white(nblock, p, &(capwhite->subwhite[i]))){
                        keep = 1;
                        if (capwhite->subwhite[i].line_with_block>
                            capwhite->line_with_block)
                            capwhite->line_with_block =
                                capwhite->subwhite[i].line_with_block;
                    }
                }
                if (keep == 0) {
                    for (i = 0; i < capwhite->numsubwhite; i++){
                        free_pair(capwhite->subwhite[i].pairright,
                                  capwhite->subwhite[i].pairleft,
                                  capwhite->subwhite[i].upper_y,
                                  capwhite->subwhite[i].lower_y);
                    }
                    capwhite->pairright = capwhite->pairleft = NULL;
                    free((struct whiteblock *)capwhite->subwhite);
                }

            } else if (block_in_white(nblock, p, capwhite))
                keep = 1;

        if (keep) {
            nblock->group++;
        }
        capwhite->index = nblock->group;
        capwhite = capwhite->next;
    }
    else {
        tmp = capwhite->next;
        remove_white(nblock, capwhite);
        free_white(capwhite);
        capwhite = tmp;
    }
}
/*
for (tmpi= nblock->firstnblock; tmpi != NULL; tmpi = tmpi->next)
    printf("index = %d, in_group = %d\n", tmpi->index, tmpi->in_group);
*/
}

/*****************************************************************************/
/*********  search block in the zone specified                       *********/
/*****************************************************************************/

void block_in_range(p, x1, y1, x2, y2, nexnblock)
unsigned char **p;
int x1, y1, x2, y2;
struct block_rect *nexnblock;
{
    int
    struct block_rect *b;

    for (i = y1; i <= y2 ; i++)
        for (j = x1; j <= x2; ){
            if (p[i][j] == ONE){
                if ((p[i][j] == ZERO)||(p[i][j] == LEFTRIGHT_ONE)||
                    (p[i][j] == LEFT_ONE)){
                    j++;
                    continue;
                }
                if (p[i][j] == ONE){
                b = (struct block_rect *)search_block(headblock);
                b->index = nexnblock->numnblock;
                search_one(p, j, i, x1, y1, x2, y2, b);
/*
printf("index = %d x1 = %d y1 = %d i = %d j = %d \n",
    b->index, b->length, b->width, b->left_x, b->upper_y, i, j);
*/
                for ( ; (j <= x2)&&(p[i][j]!=0x00); j--);
                continue;
            }
            if (p[i][j] == RIGHT_ONE){
            gain = 1;
            j--;
            continue;
            }
        } else {
            if ((p[i][j] == ZERO)||(p[i][j] == ONE)||(p[i][j] == LEFTRIGHT_ONE)||(p[i][j] == RIGHT_ONE)) {
                (p[i][j] == LEFT_ONE)){
                j++;
                continue;
            }
            if (p[i][j] == LEFT_ONE){
            gain = 0;
            j++;
            continue;
            }
        }
}
```

【図75】

```
/* printf("get out of block in recgnln"); */
}

/*****************************************************************/
/*******  count the density inside the block              *******/
/*****************************************************************/
void sort_for_blet(block)
struct block_fact *block;
{

    int i;

    if (block->if_sorted == 0){
        for (i = block->lower_y; i<= block->upper_y; i++)
            sort_x(block->pairright[i], block->pairleft[i]);
        block->if_sorted = 1;
    }
}

/*****************************************************************/
/*******  filled area  for the group in a big block        *******/
/*****************************************************************/
struct block_fact *subset;
float *area_filled;
{
    struct whiteblock *topwhite;
    int  top, bottom, left, right;
    int  x, y, pair;
    unsigned char **square;

    top = nblock->lower_y;
    bottom = nblock->upper_y;
    left = nblock->left_x;
    right = nblock->right_x;
    for (topwhite = nblock->firstwhite; topwhite != NULL;
                                topwhite = topwhite->next){
        if (topwhite->left_x < left) left = topwhite->left_x;
        if (topwhite->right_x > right) right = topwhite->right_x;
        if (topwhite->upper_y < top) top = topwhite->upper_y;
        if (topwhite->lower_y > bottom) bottom = topwhite->lower_y;
    }
/*  printf("top = %d lower = %d left=%d right=%d \n", top, bottom, left, right);
*/
    square = cmatrix(top, bottom, left, right);

    for (topwhite = nblock->firstwhite; topwhite != NULL;
                                topwhite = topwhite->next){
        for (y = topwhite->upper_y; y <= topwhite->lower_y; y++){
            for (x = topwhite->left_x; x <= topwhite->right_x; x++)
                square[y][x] = 1;
        }
    }

    for (pair = 0, y = top; y <= bottom; y++)
        for (x = left; x <= right; x++)
            if (square[y][x] == 1)
                pair++;

    *area_filled = ((float)pair/((float)(bottom-top+1)*(right-left+1)));
    free_cmatrix(square, top, bottom, left, right);
}

/*****************************************************************/
/*******  count continuous one and zero                   *******/
/*****************************************************************/
void cont_one_zero(nblock, p, avg_one, std_one, avg_zero, std_zero)
struct block_fact *nblock;
unsigned char **p;
float *avg_one, *std_one, *avg_zero, *std_zero;
{

    int i, j, k;
    int one_count, zero_count;
    int *one, *zero;
    int no_one, no_zero;
/*
printf("place 3\n");
*/
    one = ivector(0, nblock->length*nblock->width-1);
    zero = ivector(0, nblock->length*nblock->width-1);
    no_one = 0;     no_zero = 0;

    for (i = nblock->upper_y; i <= nblock->lower_y; i++){
        one_count = 0;
        zero_count = 0;
        for (j = 1; j <= nblock->pairright[i][0]; j++){
            for (k = nblock->pairleft[i][j]; k<= nblock->pairright[i][j]; k++){
                if (p[i][k]==0) {
                    one_count++;
                    if (zero_count != 0){
                        zero[no_zero++] = zero_count;
                        zero_count = 0;
                    }
                } else {
                    zero_counter;
                    if (one_count != 0) {
                        one[no_one++] = one_count;
                        one_count = 0;
                    }
                }
            }
        }
        if (one_count != 0) {
            one[no_one++] = one_count;
            one_count = 0;
        }
    }
/*
printf("no_one = %d no_zero = %d\n", no_one, no_zero);
*/
    *avg_one = *avg_zero = *std_zero = *std_one = 0;
    if (no_one > 0) avg_std(one, no_one, avg_one, std_one);
    if (no_zero > 0) avg_std(zero, no_zero, avg_zero, std_zero);
/*
printf("avg_one=%f std_one=%f avg_zero=%f std_zero=%f  \n",
    *avg_one, *std_one, *avg_zero, *std_zero);
*/
    free_ivector(one, 0, nblock->length*nblock->width-1);
    free_ivector(zero, 0, nblock->length*nblock->width-1);
}

/*****************************************************************/
/*******  count continuous one and zero                   *******/
/*****************************************************************/
```

【図76】

```
void cont_over_and_unders(nblock, p, avg_one, std_one, avg_zero, std_zero, no_over, no_under)
struct block_rect *nblock;
unsigned char **p;
float *avg_one, *avg_zero, *std_zero;
int   *no_over;
{

    int i, j, k;
    int one_count, zero_count;
    int *one, *zero;
    int no_one, no_zero;

/*
printf("place 4\n");
*/
    one = ivector(0, nblock->length*nblock->width-1);
    zero = ivector(0, nblock->length*nblock->width-1);
    no_one = 0;
    no_zero = 0;
    *no_over = 0;
    for (i = nblock->upper_y; i <= nblock->lower_y; i++){
        one_count = 0;
        zero_count = 0;
        for (j = 1; j <= nblock->pairtight[i][0]; j++){
            for (k = nblock->pairtight[i][j];
                 k <= nblock->pairleft[i][j]; k++){
                if (p[i][k]&0x80) {
                    one_count++;
                    if (zero_count != 0){
                        zero[zero_count] = zero_count;
                        zero_count = 0;
                    }
                } else {
                    zero_count++;
                    if (one_count != 0) {
                        one[no_one++] = one_count;
                        if (one_count > MAXPOINT/SCALE_RATIO) (*no_over)++;
                        one_count = 0;
                    }
                }
            }
        }
        if (one_count != 0) {
            one[no_one++] = one_count;
            if (one_count > MAXPOINT/SCALE_RATIO) (*no_over)++;
            one_count = 0;
        }
    }
/*
printf("%d %d\n", no_one, no_zero);
*/
    *avg_one = *std_zero = *std_one = 0;
    if (no_one > 0) avg_one(one, no_one, avg_one, std_one);
    if (no_zero > 0) avg_std(zero, no_zero, avg_zero, std_zero);
/*
printf("avg_one=%f std_one=%f avg_zero=%f std_zero=%f no_over = %d \n",
    *avg_one, *std_one, *avg_zero, *std_zero, *no_over);
*/
    free_ivector(one, 0, nblock->length*nblock->width-1);
    free_ivector(zero, 0, nblock->length*nblock->width-1);
}

/*****************************************************************/
/*********  sort block                                           */
/*****************************************************************/

void sort_block(headblock)
    struct block_rect *headblock;
{
    struct block_rect *block_y;
    struct block_rect *tmp;
    int i, j, num;
    int change;

    for (tmp = headblock->firstnumblock, num = 0; tmp != NULL;
                                        tmp = tmp->next, num++ );

    if (num == 0) return;

    if ((block_y = (struct block_rect**)
                malloc(num*sizeof(struct block_rect*))) == NULL){
        printf("Allocatio error! in sort_block\n");
        exit(1);
    }

    for (tmp = headblock->firstnumblock, i = 0; tmp != NULL; tmp = tmp->next)
        block_y[i++] = tmp;

    for ( i = 0; i < num-1 ; i++){
        for (j = 0, change = 0; j < num -i -1; j++)
            if (block_y[j]->upper_y > block_y[j+1]->upper_y){
                change = 1;
                tmp = block_y[j+1];
                block_y[j+1] = block_y[j];
                block_y[j] = tmp;
            }
        if (change == 0) break;
    }
/*
printf("in sort_block i = %d \n", i);
*/
    headblock->firstnumblock = block_y[0];
    block_y[0]->next = block_y[1];
    for (i = 1; i < num-1; i++){
        block_y[i]->previous = block_y[i-1];
        block_y[i]->next = block_y[i+1];
    }
    block_y[num-1]->previous = block_y[num-2];
    block_y[num-1]->next = NULL;
    headblock->numberofnumblock = block_y[num-1];
    free((struct block_rect **)block_y);
}

/*****************************************************************/
/*********  discrimination between line drwing and gray-scale picture  *********/
/*****************************************************************/

void pic_gray_discrimination(nblock, p, black_per, black_density)
struct block_rect *nblock;
unsigned char **p;
float black_per, black_density;
{
    float avg_one, std_one, avg_zero, std_zero;

/*
printf("get into pic_gray_disc\n");
*/
    cont_one_zero(nblock, p, &avg_one, &std_one, &avg_zero, &std_zero);

    if (avg_zero == 0.0){
        if (black_per > 0.3 )
            nblock->pat |= PICTURE;
        else
```

【図77】

```
    nblock->att |= PICTURE;
  } else if (avg_zero > avg_one)
      nblock->att |= PICTURE;
    else if (black_density > 0.5)
      nblock->att |= MPICTURE;
    else if (black_per > 0.3)
      nblock->att |= MPICTURE;
    else
      nblock->att |= PICTURE;

  /*
  if ((black_per < 0.3)||(avg_zero > avg_one))
    nblock->att |= PICTURE;
  else
    nblock->att |= MPICTURE;
  */

  if (black_per >= 0.5) nblock->att |= MPICTURE;
  else nblock->att |= PICTURE;

  /*
  printf("att = %x\n", nblock->att);
  printf("out of  pic_grey_dash\n");
  */
}

/*****************************************************************/
/*********  Search block in the block specified  *****************/
/*****************************************************************/

void block_in_block(p, nblock, first)
unsigned char **p;
struct block_rect *nblock, *first;
{
    int  i, j, x, getin, n, total_one, total_zero, no_over;
    float avg_one, avg_zero, sd_zero;
    float area_filled, black_per, black_density;
    struct block_rect *top;

  /*
  printf("\n\n\n*****************************\n");
  printf("get into block \n x = %d y1 - %d y2 = %d  i - %d y = %d\n",
                nblock->left_x, nblock->upper_y, nblock->lower_y,
                nblock->length, nblock->width);
  */
    if (nblock->length <= 2){
      nblock->att |= NSOLID;
  /*
  printf("att = %x\n", nblock->att);
  printf("\n*****************************\n\n");
  */
      return;
    }
    if (nblock->width <= 2){
      nblock->att |= VSOLID;
  /*
  printf("att = %x\n", nblock->att);
  printf("\n*****************************\n\n");
  */
      return;
    }

  sort_for_block(nblock);

  total_one = 0;
  total_zero = 0;
  for (i = nblock->upper_y; i <= nblock->lower_y; i++){
    for (j = 1; j <= nblock->pairright[i][0]; j++){
      for (x = nblock->pairright[i][j]; getin = 0;
                                  x <= nblock->pairleft[i][j]; x++){
        if (p[i][(BIGNUM) ] {
                                    total_zero++;
                                  } else {
                                    total_one++;
                                    if (p[i][x] == RIGHT_ZERO)&&(getin == 0)){
                                      getin = 1;
                                    } else if ((p[i][x] == LEFT_ZERO)&&(getin == 1)){
                                      getin = 0;
                                    } else if ((p[i][x] == ZERO)&&(getin == 0)){
                                      if (black_density < 0.5){
                                        block_density++;
                                        nblock->group[block_density] =
                                                                  nblock->group;
                                      }
                                      for (x++; (x < nblock->pairleft[i][j])
                                                  &&(p[i][(x)&0x07]); x++);
                                      x--;
                                      continue;
                                    }
      }
    }
  }
  black_density = ((float)total_one/((float)(total_one+total_zero));
  black_per = ((float)total_one/(float)(nblock->length*nblock->width));
  printf("group = %d black_per = %f black_density=%f \n", nblock->group, black_per,
                                                        black_density);
  /*
      if (nblock->group >= MAX_GROUP){
        fill_white_into_rect(nblock, &area_filled);
  printf("area_filled = %f \n", area_filled);
  */
        if (area_filled >= NOSPACE_SPACE)
          if ((((nblock->attr)&PICTURE)) nblock->att |= PICTURE;
          if (!((nblock->attr)&PICTURE))
            pic_grey_discrimination(nblock, p, black_per, black_density);
        } else if (((nblock->length > SOLIDSPACE)||(black_per >= 0.5))(
                  if (((nblock->length > SOLIDSPACE/SCALE_RATIO)
                      &&(nblock->width > SOLIDSPACE/SCALE_RATIO))
                        pic_grey_discrimination(nblock, p, black_per, black_density);
                  else if (((nblock->width > NOSPACE_SPACE/SCALE_RATIO)(
                      count_over_and_center(nblock, p, &avg_one, &sd_one,
                                            &avg_zero, &sd_zero, &no_over);
                      if (no_over > 0)
                      {
                        if (((avg_one > avg_zero)&&(black_per >= 0.5))||
                            (avg_one > avg_zero*2.0))
                          nblock->att |= MPICTURE;
                        else nblock->att |= PICTURE;
                      }
                  }
  /*
  if ((nblock->att != UNKNOWN)&&(nblock->att!=UNKNOWN))
    nblock->att = UNKNOWN;
  */
}
```

【図78】

```
    if ((nblock->att == FRAME)||(nblock->att==TABLE)){
/*
printf("before search block in block \n");
*/
        search_block_in_block(nblock, 0);
/*
printf("after search block in block \n");
*/
        if (!nblock->firstsubblock != NULL)&&(nblock->group >= 2))
            sort_block(nblock);

/*
for (tmp = nblock->firstsubblock; tmp != NULL; tmp = tmp->next)
    printf("index = %d id_group = %d \n", tmp->index, tmp->in_group);
*/
        rearrange_rectblock(nblock);
    }


/*
for (tmp = nblock->firstsubblock; tmp != NULL; tmp = tmp->next)
    printf("index = %d \n", tmp->index, tmp->in_group);
*/
/*
printf("att = %s\n", nblock->att);
printf("pos.....................\n\n");
*/
    if(nblock->pairright != NULL) free_pair(nblock->pairright, nblock->pairleft);printf("get out of  i_text_distribute \n"); */
        return num;
    }
/* printf("get out of  block in block \n");*/
}

/*.....                                                    .....*/
/*.....  Delete some block from the head block                .....*/
/*.....                                                    .....*/
void delete_rectblock(first, headblock)
struct block_rect *first, *headblock;
{
    remove_textblock(first, headblock);
    headblock->summblock--;
    free_rectblock(first);
}

/*.....                                                    .....*/
/*.....  find line text discriminatism                      .....*/
/*.....                                                    .....*/
int i_text_discriminate(headblock)
struct block_rect *headblock;
{
    int     num = 0;
    long    templength = 0;
    float   temp_length;
    struct block_rect *tmp, *tmpb;

/*printf("get into i_text_discriminate \n");*/
    temp_length = (float)i10.0/SCALE_RATIO;
    for (tmp = headblock->firstsubblock; tmp != NULL; ){
        if (((float)tmp->length > temp_length ){
            templength = tmp->length;
            tmpb = tmp->next;
            transfer_block(TEXT, CONTEXT, tmp, headblock, headblock);
            tmp = tmpb;
        }
        else {
            templength += (long)tmp->length;
            num++;
            tmp->att = TEXT;
            tmp = tmp->next;
        }
    }
    if (num > 0)
        headblock->avg_height = (float)templength/(float)num;
    else headblock->avg_height = 0.0;
        return num;
}

/*.....                                                    .....*/
/*.....  split a block                                      .....*/
/*.....                                                    .....*/
void block_split(head, tmp, gap1, gap2)
struct block_rect *tmp, *head;
int     gap1, gap2;
{
    struct block_rect *newblock, *b, *tmpb;
    int   i, j;

printf("get into block_split x = %d y = %d cover = %d first = %d first + %d\n",
            tmp->left_x, tmp->upper_y, tmp->cover, tmp->firstsubblock,
            tmp->firstsubblock + tmp->cover);
    newblock = new_block(3,0);

    b = tmp->firstsubblock;

    newblock->pairright = b->pairright;
    newblock->pairleft = b->pairleft;
    newblock->upper_y = gap1;
    newblock->lower_y = b->lower_y;
    newblock->right_x = b->right_x;
    newblock->left_x = b->left_x;
    for (;b != newblock->upper_y || b<newblock->lower_y; b++)
        for (i = 1; i <= b->pairleft[0][0]; i++)
            if (newblock->right_x > b->pairleft[2][i])
                for (j = newblock->pairleft[0][0]+1; j > i; j--){
                    b->pairleft[2][j] = b->pairleft[2][j-1];
                    b->pairleft[3][j] = b->pairleft[3][j-1];
                }
    for (;b != newblock->lower_y; b++)
        for (i = 1; i <= b->pairright[0][0]; i++)
            if (newblock->left_x < b->pairright[2][i])
                for (j = newblock->pairright[0][0]+1; j > i; j--){
                    b->pairright[2][j] = b->pairright[2][j-1];
                    b->pairright[3][j] = b->pairright[3][j-1];
                }
    newblock->length = newblock->lower_y-newblock->upper_y;
    newblock->width = newblock->right_x-newblock->left_x;
    newblock->sum_point = b->sum_point+newblock->length/b->length
    newblock->y = gap1;
```

【図79】

```
newblock->x = newblock->right_x;
for (i = 1; i <= b->pairright[newblock->upper_y][0]; i++)
    if (b->pairright[newblock->upper_y][i] < newblock->x)
        newblock->x = b->pairright[newblock->upper_y][i];

newblock->in_group = b->in_group;

b->lower_y = gap;
b->length = b->lower_y-b->upper_y+1;
b->on_point = b->on_point-newblock->on_point;

add_rectblock(TEXT, newblock, head);
/* printf("newblock->previous->index = %d \n", newblock->previous->index); */

    for (b = b->next; b != NULL; ){
/*
printf("b = %d b->y1 = %d b->y2 = %d gap = %d \n",
    b->index, b->upper_y, b->lower_y, tmp->firstsubblock->index,
    tmp->firstsubblock->upper_y, tmp->firstsubblock->lower_y);
*/
    if (cover_range(b->upper_y, b->lower_y, tmp->firstsubblock->upper_y,
            tmp->firstsubblock->lower_y) < 0.5){
        tmpb = b->next;
        convert_to_block(b, newblock);
        newblock->cover = COVER;
        transfer_block(TEXT, TEXT, b, tmp, newblock);
        b = tmpb;
    } else b = b->next;
    }

if (tmp->firstsubblock->next == NULL){
    tmp->firstsubblock->index = newblock->index;
    transfer_block(TEXT, TEXT, tmp->firstsubblock, tmp, head);
    remove_from_block(tmp, head);
} else {
        tmp->upper_y = tmp->firstsubblock->upper_y;
        tmp->lower_y = tmp->firstsubblock->lower_y;
        tmp->left_x = tmp->firstsubblock->left_x;
        tmp->right_x = tmp->firstsubblock->right_x;
        for (b = tmp->firstsubblock->next; b != NULL; b = b->next){
            if (tmp->upper_y > b->upper_y)
                tmp->upper_y = b->upper_y;
            if (tmp->lower_y < b->lower_y)
                tmp->lower_y = b->lower_y;
            if (tmp->left_x > b->left_x)
                tmp->left_x = b->left_x;
            if (tmp->right_x < b->right_x)
                tmp->right_x = b->right_x;
        }
        tmp->width = tmp->right_x-tmp->left_x+1;
        tmp->length = tmp->lower_y - tmp->upper_y+1;
    }

/* printf("get out of block_split\n"); */
}

/*****************************************************************/
/*                                                               */
/*      split a block                                            */
/*                                                               */
/*****************************************************************/

void block_split(head, tmp, gap1, gap2)
struct block_rect *tmp, *head;
int   gap1, gap2;
{
    struct block_rect *newblock, *b;
    int   i, y;
/*
printf("get into block_split x = %d y = %d cover = %d first = %d first.nt = %d\n",
        tmp->left_x, tmp->upper_y, tmp->cover, tmp->firstsubblock);
        tmp->firstsubblock);
*/
    newblock = new_block(0,0);

    newblock->pairright = tmp->pairright;
    newblock->pairleft = tmp->pairleft;
    newblock->upper_y = gap1;
    newblock->lower_y = tmp->lower_y;

    newblock->right_x = tmp->left_x;
    newblock->left_x = tmp->pairright[y][i];
    for (y = newblock->upper_y; y <= newblock->lower_y; y++)
        for (i = 1; i <= tmp->pairright[y][0]; i++)
            if (newblock->right_x < tmp->pairleft[y][i])
                newblock->right_x = tmp->pairleft[y][i];
    for (y = newblock->upper_y; y <= newblock->lower_y; y++)
        for (i = 1; i <= tmp->pairleft[y][0]; i++)
            if (newblock->left_x > tmp->pairright[y][i])
                newblock->left_x = tmp->pairright[y][i];

    newblock->length = newblock->lower_y-newblock->upper_y+1;
    newblock->width = newblock->right_x-newblock->left_x+1;
    newblock->on_point = tmp->on_point-newblock->length/tmp->length;

    newblock->y = gap2;

    newblock->x = newblock->right_x;
    for (i = 1; i <= tmp->pairright[newblock->upper_y][0]; i++)
        if (tmp->pairright[newblock->upper_y][i] < newblock->x)
            newblock->x = tmp->pairright[newblock->upper_y][i];

    newblock->in_group = tmp->in_group;
    newblock->cover = tmp->cover;

    tmp->lower_y = gap1;
    tmp->length = tmp->lower_y-tmp->upper_y+1;
    tmp->on_point = tmp->on_point-newblock->on_point;

    add_rectblock(TEXT, newblock, head);

/*
printf("newblock->index = %d tmp->index = %d\n", newblock->index, tmp->index);
*/

/*  printf("get out of block_split\n"); */
}

/*****************************************************************/
/*                                                               */
/*      test block_split                                         */
/*                                                               */
/*****************************************************************/

int test_split(headblock, tmp, len)
struct block_rect *headblock, *tmp;
float  len;
{
    int   i, j, *temi, split;
    int   upper, lower, left, right;
    int   gap1, gap2;

/*
printf("get into test_split len = %f \n", len);
```

【図80】

```
printf("index = %d y=%d xsid len = %f tmp->length = %d \n",
tmp->index, tmp->upper_y, tmp->left_x, len, tmp->length);
*/
        split = 0;

        if (((float)tmp->length > len*TEXT_SCALE)||
            ((float)tmp->length <= len*(1/TEXT_SCALE))){

            UPPER = (tmp->upper_y-1)*reduce_factor;
            lower = tmp->lower_y*reduce_factor;
            left = (tmp->left_x-1)*reduce_factor;
            right = tmp->right_x*reduce_factor-1;
/*
printf("place %\n");
*/
            ccol = lwater(upper, lower);
            for (i = upper; i <= lower; i++){
                total[i - upper] = 0;
                for (j = left; j <= right; j++)
                    if (original_p[i][j]!=back) total[i]++;
/*  printf("i = %d ccol[i] = %d \n", i, total[i]);  */
            }
            for (i = upper; (total[i] <= CAPPING(i <= lower); i++)
            for (j = i; j <= lower; j++){
                if (total[j] <= CAPP[
                    for (gap1 = j; (total[j] <= CAPPING(j <= lower);gap2 = j; j++);
                    if (gap1 == lower) break;
                    gap1 = (gap1/reduce_factor)+1;
                    gap2 = (gap2/reduce_factor)+1;
                    if (gap1 == gap2){
                        if (gap1 == tmp->upper_y) gap1 = tmp->upper_y+1;
                        else if (gap2 == tmp->lower_y) gap2 = tmp->lower_y-1;
                            else gap1 = gap2+1;
                    }
/*
  printf("gap 1= %d gap 2 = %d y = %d \n", gap1, gap2, tmp->left_x,
                        tmp->upper_y);
*/
                    if (tmp->firstsubblock != NULL)
                    block_split(headblock, tmp, gap1, gap2);
                    else
                        block_split(headblock, tmp, gap1, gap2);
                    split = 1;
                    break;
                }
            }
            free_vector(ccol, upper, lower);
        }
/* printf("got out of text_split\n");    */
        return split;
}

/*****************************************************************************/
/*********  Second time text discrimination                          *********/
/*****************************************************************************/

void s_text_discriminate(headblock)
struct block_rect *headblock;
{
    int    min = 0;
    long   templength = 0;
    float  temp_length;
    struct block_rect *tmp, *tmp1;

/*printf("get into s_text_discriminate \n");*/
    temp_length = headblock->avg_height;

    if (reduce_factor > 1)
        for (tmp = headblock->firstsubblock; tmp != NULL; tmp = tmp->next)
            text_split(headblock, tmp, temp_length);


    for (tmp = headblock->firstsubblock; tmp != NULL; ){
        if (((float)tmp->length > temp_length*TEXT_SCALE){
            tmp->att = UNKNOWN;
            if (tmp->cover == COVER) {
                for (tmp1 = tmp->firstsubblock; tmp1 != NULL; tmp1 = tmp1->next)
                    if (tmp->cover == OBJECT) {
                        transfer_block(TEXT, NONTEXT, tmp, tmp1, tmp);
                        break;
                    }
            }
            tmp1 = tmp->next;
            transfer_block(TEXT, NONTEXT, tmp, headblock, headblock);
            tmp = tmp1;
        } else {
            templength += (long)tmp->length;
            num++;
            tmp->att = TEXT;
            tmp = tmp->next;
        }
    }
    if (num > 0)
    headblock->avg_height = (float)templength/(float)num;
    else
    headblock->avg_height = 0.0;
    if (total_text < num /(  text_length = headblock->avg_height;
                             total_text = num;
                          )

/* printf("num = %d text_length = %f \n", num, text_length); */
/*printf("got out of  s_text_discriminate \n");*/
}

/*****************************************************************************/
/*********  Discriminate text and context                            *********/
/*****************************************************************************/

void text_discriminate(headblock)
struct block_rect *headblock;
{
    struct block_rect *tmp;

    i_text_discriminate(headblock);
    s_text_discriminate(headblock);
/*
    if (i_text_discriminate(headblock))
    else {
        for(tmp = headblock->firstsubblock;
                            tmp != NULL; tmp = tmp->next)
            text_discriminate(tmp);
    }
*/
}

/*****************************************************************************/
/*********  Count the density inside the block                       *********/
/*****************************************************************************/
```

【図81】

```
                                                    i <=block->lower_y; i++){
                for (j = block->left_x; j <= block->right_x; j++)
                    if (p[i][j]&0x01)
                        sigma += ((float)((j-block->left_x)*
                                                    (j-block->left_x)));
                }
                sigma /= block->length;
                if (sigma <= SOLIDTHRD)
                    return VSOLID|UP;
                else
                    return UNKNOWN;
                }
            }
}

/*************************************************************************/
/******** Frame testing                                          ********/
/*************************************************************************/

void frame_discriminate(p, block)
unsigned char **p;
struct block_rect    *block;
{
        int  num, w, i, longest, j, templong, t;
        long sigma;

/*
printf("get into frame_dis \n");
*/
        sort_for_block(block);

        if ((block->length > 150/SCALE_RATIO)&&(block->width > 300/SCALE_RATIO)){
           t = (int)(SINCDEGREE*(float)block->length+3.0*STANDARDFONT);
           w = (block->width-t) > 0 ? block->width - t: block->width;

           for (num=0, i=block->upper_y+1, sigma = 0.0;
                                        i <= block->lower_y-1; i++){
               for (longest = 0, j = 1; j <= block->pairright[i][0]; j++)
                   if ((templong=block->split:ct[i][j]-block->pairright[i][j+1]
                                                        >longest)
                       longest = templong;
               if (longest > w){
                  sigma -=(long)(block->width-longest);
                  num++;
               }
           }
           if (num > 0) sigma /= num;

           /*  frame test and content test  */

           if ((num >= (int)((float)(block->length-2)*0.9))&&(sigma <= t*2)){
               block->att = FRAME;
        } else if (((float)block->length < 150/SCALE_RATIO)&&
                                        ((float)block->width > 300/SCALE_RATIO))
               block->att = if_hasild(p, block);
           else if (((float)block->width < 150/SCALE_RATIO)&&
                                        ((float)block->length > 300/SCALE_RATIO))
               block->att = if_vsolid(p, block);
/* printf("att = %x \n",block->att); */
 }

/*************************************************************************/
/******** Recover the covered block into original head block     ********/
/*************************************************************************/

void text_recover(head, first)
struct block_rect  *head, *first;
{
    struct block_rect *tmp, *tmpnext;
    float             tmpl;

    tmpl = (head->avg_height == 0.0) ? text_length :head->avg_height;

    for (tmp = first->firstsubblock; tmp!=NULL; ){
        if (tmp->cover == COVERED)
            {
                if (!((float)tmp->length <= tmpl*TEXT_SCALE)
                                        &&((tmp->width/w->head->width/WIDTH_RATIO))){
                   tmpnext = tmp->next;
                   transfer_block(TEXT, TEXT, tmp, first, head);
                   tmp = tmpnext;
                }
                else{
                      if (tmp->length > tmpl*TEXT_SCALE)
                         if (text_split(first, tmp, tmpl)) continue;
                      tmpnext = tmp->next;
                      transfer_block(TEXT, MONTEXT, tmp, first, head);
                      tmp = tmpnext;
                   }
            } else  tmp = tmp->next;
    }
}

/*************************************************************************/
/******** Recover the covered montext block into original head block     ********/
/*************************************************************************/

void mtext_recover(head, first)
struct block_rect  *head, *first;
{

    struct block_rect *tmp, *tmpnext;
    float             tmpl;

    tmpl = (head->avg_height == 0.0) ? text_length :head->avg_height;

    for (tmp = first->firstmontextblock; tmp!=NULL; ){
        if (tmp->cover == COVERED)
            {
               tmpnext = tmp->next;
               if (!((float)tmp->length <= tmpl*TEXT_SCALE)
                                        &&(tmp->width/w->head->width/WIDTH_RATIO)){
                   tmpnext = tmp->next;
                   transfer_block(TEXT, TEXT, tmp, first, head);
                   tmp = tmpnext;
               }
               else {
                     if (tmp->length > tmpl*TEXT_SCALE)
                        if(text_split(first, tmp, tmpl)) continue;
                     tmpnext = tmp->next;
                     transfer_block(TEXT, MONTEXT, tmp, first, head);
                     tmp = tmpnext;
                    }
            } else tmp = tmp->next;
    }
}

/*************************************************************************/
/******** Processing for one montext block contained             ********/
/*************************************************************************/

void one_montext_contained(block, head)
```

```
struct block_rect *block, *head;
{
    struct block_rect *tmp;
/*
printf("get into one nontextblock \n");
printf("cove = %d firstsubblock = %d firstnon = %d currentnon=%d \n",
       block->cover, block->firstsubblock, block->firstnontextblock,
       block->currentnontextblock);
*/
    if ((block->cover == COVER)&&(block->firstsubblock == NULL)&&
        (block->firstnontextblock->cover == ORIGINAL)){

       block->firstnontextblock->next = block->next;
       if (head->firstnontextblock == block){
          head->firstnontextblock = block->firstnontextblock;
          if (head->currentnontextblock == block)
             head->currentnontextblock = block->firstnontextblock;
          else
             block->next->previous = block->firstnontextblock;
       } else {
                if (head->currentnontextblock == block)
                   head->currentnontextblock = block->firstnontextblock;
                else
                   block->next->previous = block->firstnontextblock;
                block->previous->next = block->firstnontextblock;
                block->firstnontextblock->previous = block->previous;
             }
       block->firstnontextblock->cover = 0;
       block->firstnontextblock->index = block->index;
/*     block->firstnontextblock->cover = 0; */
       free((struct block_rect *)block);

    }
/* printf("get out of  one nontextblock \n"); */
}


/*****************************************************************************/
/******** Classify nontext block                                     ********/
/*****************************************************************************/


void nontext_classify(p, tmpb, head)
unsigned char **p;
struct block_rect    *tmpb, *head;
{
       struct block_rect  *tmpsub;

/*  printf("get into nontext_class\n");  */
          if (tmpb->att != UNKNOWN) return;
          if (tmpb->firstnontextblock != NULL)
             for ( tmpsub = tmpb->firstnontextblock; tmpsub != NULL;
                                                  tmpsub = tmpsub->next) {
                frame_discriminate(p, tmpsub);
                if (((tmpsub->att==(HSOLID|DOWN))||(tmpsub->att == (HSOLID|UP)))&&
                    (tmpb->cover == COVER)&&(tmpsub->cover == ORIGINAL))
                   {
                     text_recover(head, tmpb);
                     ntext_recover(head, tmpb);
/*                   one_nontext_contained(tmpb);  */
                   }
             }
          else{
            frame_discriminate(p, tmpb);
          }
/*  printf("get out nontext_class\n");  */
}


/*****************************************************************************/
/******** Classify nontext block                                     ********/
/*****************************************************************************/

void nt_class(p, tmpb)
unsigned char **p;
struct block_rect *tmpb;
{
       struct block_rect *subb;

       for (subb = tmpb->firstnontextblock; subb != NULL; subb = subb->next)
          nontext_classify(p, subb, tmpb);

}

/*****************************************************************************/
/******** Halftone testing                                           ********/
/*****************************************************************************/

int if_halftone(block,  head)
struct block_rect *block, *head;
{
       int         halft = 0;
       int         num_t, dot;
       float       tmpl;
       struct block_rect *tmp, *tmpb;


/* printf("get into if_halftone, x = %d y = %d\n", block->left_x, block->upper_y); */
       if (block->firstsubblock == NULL) return halft;
       else {
              for (num_t = 0, dot = 0, tmp = block->firstsubblock;
                                       tmp!= NULL; tmp = tmp->next){
/* printf("inde %d  %d y = %d l = %d w = %d \n",
       tmp->index, tmp->left_x, tmp->upper_y,  tmp->length, tmp->width); */
                 if ((((float)tmp->length < MPOINT/SCALE_RATIO)&&
                      (((float)tmp->width < MPOINT/SCALE_RATIO))
                   dot++;
                 num_t++;
              }
       }
/* printf("dot = %d num = %d \n", dot, num_t);*/
       if ((dot >= MDOT) &&(dot >= ((num_t+1)/2)))
                                       /* ?? need more consideration */
         halft = MPICTURE;

/*
       if ((halft == MPICTURE)&&(block->cover == COVER)) {
         for (tmp = block->firstsubblock; tmp != NULL; )
            if (((float)tmp->length <= tmpl*TEXT_SCALE)
                                        &&(tmp->width<=head->width/WIDTH_RATIO)){
            if ((tmp->cover == COVERED)&&
                                    (tmp->length >= MPOINT/SCALE_RATIO)){
              tmpb = tmp->next;
              transfer_block(TEXT, TEXT, tmp, block, head);
              tmp = tmpb;
            } else tmp = tmp->next;
         ntext_recover(head, block);
         one_nontext_contained(block);
       }
*/
/*
```

【図84】

```
printf("out out of halftone text\n");
*/
    return half;
}

/*****************************************************************************/
/*                                                                           */
/*  Determine text inside content block                                      */
/*                                                                           */
/*****************************************************************************/

void text_in_content(first)
struct block_rect *first;
{
    struct block_rect *tmp;
/* printf("get into text in content\n"); */
    for (tmp = first; tmp != NULL; tmp = tmp->next){
        if (tmp->firstblock != NULL) text_discriminate(tmp);
    }
/*printf("out of text in content\n");*/
}

/*****************************************************************************/
/*                                                                           */
/*  Test if halftone for the content block under                             */
/*                                                                           */
/*****************************************************************************/

void halftone_test(testblock)
struct block_rect *testblock;
{
    struct block_rect *tmp;

    for (tmp = testblock->firstcontextblock; tmp != NULL; tmp = tmp->next)
        tmp->att |= if_halftone(tmp, testblock);
}

/*****************************************************************************/
/*                                                                           */
/*  Search the content of the first level nearest block                      */
/*                                                                           */
/*****************************************************************************/

void firstlevelnontextsearch(p, first)
unsigned char   *p;
struct block_rect   *first;
{
    struct block_rect *tmpb, *tmpbnext, *tmpl, *tmpnl;
    float  tmpl_ds;
    int    m, y, no_total, total, i;

/*
printf("get into firstlevelnontextsearch\n");
*/
    for (tmpb = first->firstnextsamelevel; tmpb != NULL; ){
/* printf("index = %d \n", tmpb->index); */
        content_classify(p, tmpb, first);
        if ((tmpb->firstcontextblock == NULL)&&(tmpb->firstsubblock == NULL)){
/*
printf("in firstlevel block_in_block, x = %d y = %d \n", tmpb->left_x,
            tmpb->upper_y);
*/
            if ((((tmpb->length > 130/SCALE_RATIO)||(tmpb->width > first->width/2))&&
                (tmpb->attribute&(1<<ATTRIBUTE_PICTURE)))||
                (tmpb->attribute&(1<<ATTRIBUTE_FRAME))){
                tmpb = tmpb->next;
            }else{
                tmpbnext = tmpb->next;
                if (tmpb->first_in_group == NULL){
                    tmpb->attribute |= (1<<ATTRIBUTE_PICTURE);
                    tmpb->attribute |= (1<<ATTRIBUTE_FRAME);
                }
                if (tmpb->first_in_group == NULL){
                    tmpb->attribute |= (1<<ATTRIBUTE_PICTURE);
                    tmpb->attribute |= (1<<ATTRIBUTE_FRAME);
                }
                if (tmpb->first_in_group != NULL){
                    tmpl = tmpb->first_in_group;
                    while (tmpl != NULL){
                        tmpnl = tmpl->next;
                        if (tmpl->attribute&(1<<ATTRIBUTE_FRAME)){
                            tmpb->attribute |= (1<<ATTRIBUTE_FRAME);
                            break;
                        }
                        tmpl = tmpnl;
                    }
                }
                tmpb = tmpbnext;
            }
        }else{
            tmpb = tmpb->next;
        }
    }
/*
printf("before text ds \n");
*/
/*
for (tmpl = tmpb->firstsubblock; tmpl != NULL; tmpl = tmpl->next)
    printf("index = %d, in_group = %lx", tmpl->index, tmpl->in_group);
*/
/*
printf("after text ds \n");
for (tmpl = tmpb->firstcontextblock; tmpl != NULL; tmpl = tmpl->next)
    printf("index = %d, in_group = %lx", tmpl->index, tmpl->in_group);
*/
/*
printf("get into firstlevelnontextsearch\n");
*/
    for (tmpb = first->firstnextsamelevel; tmpb != NULL; ){
/* printf("index = %d \n", tmpb->index); */
        content_classify(p, tmpb, first);
        if ((tmpb->firstcontextblock == NULL)&&(tmpb->firstsubblock == NULL)){
/*
printf("in firstlevel block_in_block, x = %d y = %d \n", tmpb->left_x,
            tmpb->upper_y);
*/
            if ((((tmpb->length > 130/SCALE_RATIO)||(tmpb->width > first->width/2))&&
                (tmpb->attribute&(1<<ATTRIBUTE_PICTURE)))||
                (tmpb->attribute&(1<<ATTRIBUTE_FRAME))){
                tmpb = tmpb->next;
            }else{
                tmpbnext = tmpb->next;
                if (tmpb->first_in_group == NULL){
                    tmpb->attribute |= (1<<ATTRIBUTE_PICTURE);
                    tmpb->attribute |= (1<<ATTRIBUTE_FRAME);
                }
                if (tmpb->first_in_group != NULL){
                    tmpl = tmpb->first_in_group;
                    while (tmpl != NULL){
                        tmpnl = tmpl->next;
                        if (tmpl->attribute&(1<<ATTRIBUTE_FRAME)){
                            tmpb->attribute |= (1<<ATTRIBUTE_FRAME);
                            break;
                        }
                        tmpl = tmpnl;
                    }
                }
                tmpb = tmpbnext;
            }
        }else{
            tmpb = tmpb->next;
        }
    }
/*
printf("in firstlevel\n");
*/
    sort_for_block(tmpb->firstcontextblock);

    total = 0;
    no_total = 0;
    for (tmpb = tmpb->firstsubblock; tmpb != NULL;
                                     tmpb = tmpb->next){
```

【図85】

```
printf("index = %d l = %d \n", tmpbb->index, tmpbb->length);
/*
                if ((((float)tmpbb->length * 150.0/SCALE_RATIO+4
                    (tmpbb->width + tmpbb->width/3.0))<
                    total == tmpbb->length)
                    no_total++;
                }
            }
            if (no_total > 0)
                tmpbb->avg_height = (float)total/(float)no_total;

            tmp1 = first->avg_height
                        ((first->avg_height
            sd = (tmpbb->avg_height >= HPOINT/SCALE_RATIO?
                (HPOINT/SCALE_RATIO-2.0):(HPOINT/SCALE_RATIO/
/*
printf("tmp1 = %f  \n", tmp1);
*/
            tmpnt = tmpb->firstnontextblock;

/*
for ( y = tmpnt->upper_y; y <= tmpnt->lower_y; y++){
    printf(" y = %d pair = %d ", y, tmpnt->pairright[y][0]);
    for (i = 1; i <= tmpnt->pairright[y][0]; i++)
        printf("%d %d ", tmpnt->pairright[y][i], tmpnt->pairleft[y][i]);
    printf("\n");
}
*/
            for (tmpbb = tmpb->firstsubblock; tmpbb!=NULL; ){
            for (rm = 1, y=tmpnt->upper_y; y <= tmpbb->lower_y; y++)
                if ( tmpnt->pairright[y][0] != 0 ){
                    if(f_overlap(tmpbb->left_x, tmpbb->right_x,
                        tmpnt->pairright[y][1],
                        tmpnt->pairleft[y][tmpnt->pairright[y][0]])){
                            rm = 0;
                            break;
                    }
                }
/*
printf("index = %d  rm = %d \n", tmpbb->index, rm);
*/
            if (rm == 1){
                if ((float)tmpbb->length >= tmp1*TEXT_SCALE){
                    if ((float)tmpbb->width >= sd){
                        tmpbb->width += first->width/WIDTH_RATIO);
                        tmpbbnext = tmpbb->next;
                        transfer_block(TEXT, TEXT, tmpbb, tmpb, first);
                        tmpbb = tmpbbnext;
                    } else tmpbb = tmpbb->next;
                } else tmpbb = tmpbb->next;
            }
            tmpbb = tmpb->next;
            one_center_contained(tmpb, first);
        }
/*
printf("after one_nontext \n");
*/
            tmpb = tmpbb;
        } else if (((tmpb->attribute)!=(tmpb->cover -- COVER))){
                free_recover(first, tmpb);
        }
/*printf("after free recc cover \n");
for (tmpb = tmpb->firstnonblock->firstsubblock; tmp1 != NULL; tmp1 = tmp1->next)
    printf("index = %d, in_group = %d\n", tmp1->index, tmp1->in_group);
*/
                block_recover(first, tmpb);
                tmpb = tmpbb;
            }
        }
/*
printf("after free\n");
*/
    }
/*
printf("get out firstlevelnontextsearch\n");*/
}

/*************************************************************************/
/******** Print block message                                    *********/
/*************************************************************************/

void print_block(level, initial_block)
int     level;
struct block_rect  *initial_block;
{
    struct block_rect* tmp;
    int nextlevel, i;

    nextlevel = level+1;
    for (tmp = initial_block->firstsubblock; tmp != NULL;tmp=tmp->next){
        for ( i = 0; i <= (level-1)*3; i++) fprintf(block_file, " ");
        fprintf(block_file,"index = %d ", tmp->index);
        fprintf(block_file,"left = %d ", tmp->left_x);
        fprintf(block_file," upper_y = %d  right_x = %d  lower_y = %d\n",
            tmp->left_x, tmp->upper_y, tmp->right_x, tmp->lower_y);
            tmp->density, tmp->on_point,
            tmp->numsubblock, tmp->cover, tmp->group, tmp->in_group);
        print_block(nextlevel, tmp);
    }
    for (tmp = initial_block->firstnonsubblock; tmp != NULL;tmp=tmp->next){
        for ( i = 0; i <= (level-1)*3; i++) fprintf(block_file, " ");
        fprintf(block_file, "nonindex = %d ", tmp->index);
        fprintf(block_file,"left = %d  right_x = %d  width = %d\n",
            tmp->left_x, tmp->upper_y, tmp->length, tmp->width,
            tmp->density, tmp->on_point, tmp->att,
            tmp->numsubblock, tmp->cover, tmp->group, tmp->in_group);
        print_block(nextlevel, tmp);
    }
}

/*************************************************************************/
/******** Release block                                          *********/
/*************************************************************************/

void release_block(initial_block)
struct block_rect *initial_block;
{
    struct block_rect *tmp1, *tmp2, *tmp11, *tmp12 ;
    int   i;

    tmp1 = initial_block->firstsubblock;
    tmp2 = initial_block->firstnonsubblock;
    for (; tmp1 != NULL; ){
        tmp11 = tmp1->next;
```

【図87】

```
/*****************************************************************/
/*****************************************************************/
/*****************************************************************/
/*****************************************************************/
/*****************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
#include "blockgeneral.h"
#include "blockline.h"

void    exit();
FILE    *fopen(), *line_file;
int     width=0;
int     text_len;
float   text_length;
extern  int  redraw_flag=1;


char *fillstr(s,len,"empty", "strlip", "text", "forwardin", "surroundout");
int  num_lower, up_number, total_num;
int  lower_no, upper_no;
unsigned char **original_s;
unsigned char **text_area;

/*****************************************************************/
/******* allocation blockpointer                         *******/
/*****************************************************************/

struct block_rect **new_blockpointer(nl, nh)
int nl, nh;
{
    struct block_rect **v;

    v = (struct block_rect **) malloc((nh-nl+1)*sizeof(struct block_rect *));
    if (!v) {
        printf("error: in the allocation of blockpointer\n");
        exit(1);
    }
    v -= nl;
    return v;
}

/*****************************************************************/
/******* sort block_rect in y order                      *******/
/*****************************************************************/

struct block_rect **sort_caseblock(numblock, headblock)
int
struct block_rect *headblock;
{
    int     i, j, step;
    struct block_rect **sap;
    struct block_rect *block_y;

    *numblock = 0;

    if (headblock->attribute){
        for (sap = headblock->firstsubblock; sap!= NULL; sap = sap->next)
            if ((((float)sap->length >= XPOINT/SCALE_RATIO)&&
                 (((float)sap->length <= text_len*2.0))&&
                 (((float)sap->width >= YPOINT/SCALE_RATIO))
                (*numblock)++;
    } else
        for (sap = headblock->firstsubblock; sap != NULL; sap = sap->next)
            (*numblock)++;

    if ((*numblock) == 0) return NULL;
    block_y = new_blockpointer(0, (*numblock)-1);

    if (headblock->attribute){
        for (sap = headblock->firstsubblock, i = 0; sap != NULL; sap = sap->next)
            if ((((float)sap->length >= XPOINT/SCALE_RATIO)&&
                 (((float)sap->length <= text_len*2.0))&&
                 (((float)sap->width >= YPOINT/SCALE_RATIO))){
                block_y[i++] = sap;
            }
    } else {
        for (sap = headblock->firstsubblock, i = 0; sap != NULL; sap = sap->next){
            block_y[i++] = sap;
        }
    }

    for (i = 0; i < *numblock; i++){
        for (j = 0, step = 0; j < *numblock -i -1; j++)
            if (block_y[j]->upper_y > block_y[j+1]->upper_y){
                sap = block_y[j+1];
                block_y[j+1] = block_y[j];
                block_y[j] = sap;
                step = 1;
            }
        if (step == 0) break;
    }

    return block_y;
}

/*****************************************************************/
/******* plot area                                       *******/
/*****************************************************************/

unsigned char **plot_area(num, s_block, nrl, nrh, ncl, nch)
int num;
struct block_rect **s_block;
int nrl, nrh, ncl, nch;
{
    int i, k, h;

    text_area = cmatrix(nrl, nrh, ncl, nch);

    for ( i = 0; i < num; i++)
        for (k = s_block[i]->upper_y; k <= s_block[i]->lower_y; k++)
            for (h = s_block[i]->left_x; h <= s_block[i]->right_x; h++)
                text_area[k][h] = 1;
}

/*****************************************************************/
/******* test if on block embrace the other, or vice versa *******/
/*****************************************************************/

int xinifirst, seqmod;
struct block_rect *first, *second;
{
    if ((second->left_x <= first->left_x)&&(second->right_x >= first->right_x))
        return 1;
```

【図88】

```
    if ((first->left_x <= second->left_x)&&(first->right_x >= second->right_x))
        return 2;

    return 0;
}

/*.....*/
/*..... test if y range overlap                                    .....*/
/*.....                                                            .....*/

int ycouch(first, second)
struct block_rect *first;
struct linablock *second;
{
    int i;

    if ((cover_range(first->upper_y, first->lower_y,
        second->upper_y, second->lower_y)>LINDOVERLAP)||
        (cover_range(second->upper_y, second->lower_y,
            first->upper_y, first->lower_y)>LINDOVERLAP))
        return 1;
    else
        return 0;
}

/*.....*/
/*..... like fillbg test                                           .....*/
/*.....                                                            .....*/

fillcondition fill_test(x, current_block, head, width)(((
int
struct block_rect *current_block, *head;
struct block_rect **width[1][1];
{

    int          i, j, intext;
    int          begin, end;
    fillcondition fill = EMPTY;
    struct block_rect *previous_block;

    begin = current_block->left_x;
    end = current_block->right_x;

    previous_block = NULL;
    for (i = begin; i <= end; previous_block = width[i][1]; i++){
        if ((width[i][1] == NULL)||(width[i][1] == previous_block)) continue;
        if ((width[i][1]->lower_y > current_block->upper_y) ||
            ((fillcond)(current_block->lower_y>=(width[i][1]->upper_y)
                                        && test_len*2)))
            {current_block->length=(int)(((float)width[i][1]->length*DOT_MOTION)&0xfffe);
            current_block->width = width[i][1]->width - DOT_WIDTH);}
            intext = xin(current_block, width[i][1]);
            if (intext == 1){
                fill = SAMEDOOR;
                break;
            } else if (intext == 2) {
                fill = MIDDLEDOOR;
                connect_head(head,current_block);
                transfer_block(head, &DOT, width[i][1], head,current_block);
                for (j = i+1; width[i][1] == width[i][1]; j++)
                    width[i][1] = NULL;
                i = j-1;
                continue;
            } else {
                fill = OVERLAP;
                continue;
            }
        else {
            fill = NEXT;
            break;
        }
    }
    *x = i;
    return fill;
}

/*.....*/
/*..... Allocate new line block                                    .....*/
/*.....                                                            .....*/

struct linablock *new_linblock(nl, sh)
int nl, sh;
{
    int i;
    struct linablock *v;

    v = (struct linablock *) malloc((unsigned)(sh-nl+1)*
                sizeof(struct linablock));

    if (!v){
        printf("allocation error: line block \n");
        exit(1);
    }
    v = v-nl;
    for (i = nl; i <=sh; i++) {
        v[i].first_block = NULL;
        v[i].currentline = v[i].firstcurrentline = NULL;
        v[i].firstcurrentline = v[i].currentcurrentline = NULL;
        v[i].previous = v[i].next = NULL;
        v[i].upper_y = v[i].lower_y = v[i].left_x = v[i].right_x = 0;
        v[i].width = v[i].length = 0;
        v[i].group = 0;
        v[i].att = UNKNOWN;
        v[i].masubblock = 0;
        v[i].index = 0;
        v[i].old_map = 0;
        v[i].under_caption = NULL;
        v[i].r_caption = v[i].l_caption = 0;  */
    }
    return v;
}

/*.....*/
/*..... Chain the nearest line block                               .....*/
/*.....                                                            .....*/

struct linablock *new_nearestline(headline)
struct linablock *headline;
{
    struct linablock *a;

    a = new_linblock(0,0);
    if (headline->firstnearestline == NULL)
        headline->firstnearestline = a;
    else{
        headline->currentnearestline->next = a;
        a->previous = headline->currentnearestline;
    }
```

【図89】

```
        headline->currentnontextline = n;
        return n;
}

/*************************************************************************/
/********  Chain the line block                                  ********/
/*************************************************************************/

struct lineblock *new_line(headline)
struct lineblock *headline;
{
        struct lineblock *n;

        n = new_lnblock(0,0);
        if (headline->firstsubline == NULL)
           headline->firstsubline = n;
        else{
              headline->currentsubline->next = n;
              n->previous = headline->currentsubline;
            }
        headline->currentsubline = n;
        return n;
}

/*************************************************************************/
/******** Add block into line block forward                      ********/
/*************************************************************************/

void fadd_block_into_line(rect_block, head_block, line_block)
struct block_rect *rect_block, *head_block;
struct lineblock  *line_block;
{
        struct block_rect *tmpblock;

        remove_textblock(rect_block, head_block);
        head_block->numsubblock--;
        if (line_block->first_block == NULL){
           line_block->first_block = line_block->current_block = rect_block;
           rect_block->previous = rect_block->next = NULL;
           line_block->left_x = rect_block->left_x;
           line_block->right_x = rect_block->right_x;
           line_block->upper_y = rect_block->upper_y;
           line_block->lower_y = rect_block->lower_y;
           line_block->numsubblock++;
        } else {
                line_block->current_block->next = rect_block;
                rect_block->previous = line_block->current_block;
                line_block->current_block = rect_block;
                rect_block->next = NULL;
                if (rect_block->left_x < line_block->left_x)
                           line_block->left_x = rect_block->left_x;
                if (rect_block->right_x > line_block->right_x)
                           line_block->right_x = rect_block->right_x;
                if (rect_block->upper_y < line_block->upper_y)
                           line_block->upper_y = rect_block->upper_y;
                if (rect_block->lower_y > line_block->lower_y)
                           line_block->lower_y = rect_block->lower_y;
                line_block->numsubblock++;
              }
}

/*************************************************************************/
/******** Add block into line block backward                     ********/
/*************************************************************************/

void badd_block_into_line(rect_block, head_block, line_block)
struct block_rect *rect_block, *head_block;
struct lineblock  *line_block;
{
        struct block_rect *tmpblock;

        remove_textblock(rect_block, head_block);
        head_block->numsubblock--;
        if (line_block->first_block == NULL){
           line_block->first_block = line_block->current_block = rect_block;
           rect_block->previous = rect_block->next = NULL;
           line_block->left_x = rect_block->left_x;
           line_block->right_x = rect_block->right_x;
           line_block->upper_y = rect_block->upper_y;
           line_block->lower_y = rect_block->lower_y;
           line_block->numsubblock++;
        } else {
                line_block->first_block->previous = rect_block;
                rect_block->next = line_block->first_block;
                line_block->first_block = rect_block;
                rect_block->previous = NULL;
                if (rect_block->left_x < line_block->left_x)
                           line_block->left_x = rect_block->left_x;
                if (rect_block->right_x > line_block->right_x)
                           line_block->right_x = rect_block->right_x;
                if (rect_block->upper_y < line_block->upper_y)
                           line_block->upper_y = rect_block->upper_y;
                if (rect_block->lower_y > line_block->lower_y)
                           line_block->lower_y = rect_block->lower_y;
                line_block->numsubblock++;
              }
}

/*************************************************************************/
/******** test if a gap between two rectblock                    ********/
/*************************************************************************/

int if_gap(block1, block2, headblock)
struct block_rect *block1, *block2, *headblock;
{
        int center_x, i, count1, count2;
        int bound1, bound2;
        int len, maxcount, x_maxcount;


        if (block2->left_x == block1->right_x+1) return 0;

        bound1 = min(block1->upper_y, block2->upper_y);
        bound2 = max(block1->lower_y, block2->lower_y);


        for (center_x = block1->right_x+1; center_x < block2->left_x; center_x++){
            for ( i = bound1, count1 = 0; i <= bound2; i++)
                if (original_p[i][center_x]&GAPMARK) {
                   printf("%x ", original_p[i][center_x]);
                   count1++;
                }

           if (count1 == (bound2-bound1+1)) return 1;
        }

        len = (MAXFONT/SCALE_RATIO < 15*text_len)? MAXFONT/SCALE_RATIO:15*text_len;
        maxcount = 0;
        x_maxcount = 0;
```

【図90】

```
for (center_x = block1->right_x+1; center_x < block2->left_x; center_x++){
    for (count = 0, i = (bound1+bound2)/2;
            i <= headblock->lower_y; i++){
        if (original_p[i][center_x]==ssi) break;
        else count++;
    }
    if (count) > maxcount) {
        maxcount = count;
        x_maxcount = center_x;
    }
}
if (maxcount >= leaf){
printf("len = %d max_count = %d x = %d y = %d \n", len, maxcount, x_maxcount,
                (bound1+bound2)/2 );
    for (i = (bound1+bound2)/2; count1 = 1; count1 <= maxcount; count1++, i--){
        original_p[i][x_maxcount] = GARBAGE;
    return 1;
    }
    else
    return 0;

    for (count = 0, i = upper; i <= lower; i++)
        if (rec_area[i][center_x]==0) count++;
printf(" x = %d upper_y = %d lower_y = %d count = %d \n",
        center_x, upper, lower, count);

    if (count <= GARBAGE)
        return 1;
    else
        return 0;
}

/*****************************************************************/
/*******  Group block into line                          *********/
/*****************************************************************/

void group_connected_block(stopx, column, currentlineblock, headblock, widthfill)
int   stopx, column;
struct linablock *currentlineblock;
struct block_rect *headblock;
struct block_rect **widthfill;
{
    int   x, count, i;
    int   first, second;
    struct block_rect *tmp_block;

    /* add the first block forward */
    add_block_into_line(widthfill[stopx], headblock, currentlineblock);
    for (x = stopx -1, tmp_block = widthfill[stopx]; x >= 0; x--){
        if (widthfill[x] == tmp_block) {
            widthfill[x] = NULL;
            continue;
        }

        count = 0;
        for (; (x >= 0)&&(count<widthset); x--, count++){
            if (widthfill[x] == NULL) continue;
            if ((ytouch(widthfill[x], currentlineblock))) {
                for ( ; (x > 0)&&(widthfill[x-1] == widthfill[x])
                        &&(count < widthset); x--, count++);
            } else break;
        }
        if ((count < widthset)&&(count > 0)){
            total_gap += count;
            gap_nonover++;
        } else
            gap_over++;
    }
    if (count >= widthset)
        current[inablock->side_gap |= LEFTMAP;
    for (i = 0;(count >= widthset[i] break;
    /* printf("index = %d \n", currentlineblock->index);*/
    if ((((gap*widthfill[x], currentlineblock->first_block, headblock))){
        &&(currentlineblock->right_x-currentlineblock >= text_len*10)) {
        currentlineblock->side_gap |= LEFTMAP;
        break;
    }
    add_block_into_line(widthfill[x], headblock, currentlineblock);
    tmp_block = widthfill[x];
    widthfill[x] = NULL;
}
for (x = stopx+1; tmp_block = widthfill[stopx]; x <= column; x--){
    if (widthfill[x] == tmp_block) {
        widthfill[x] = NULL;
        continue;
    }

    count = 0;
    for (; (x < column)&&(count<widthset); x++, count--){
        if (widthfill[x] == NULL) continue;
        if ((ytouch(widthfill[x], currentlineblock))) {
            for ( ; (x < column-1)&&(widthfill[x] == widthfill[x+1]);
                    x++, count--);
        } else break;
    }
    if ((count < widthset)&&(count >0)){
        total_gap += count;
        gap_nonover++;
    } else
        gap_over++;
    if (count >= widthset)
        current[inablock->side_gap |= RIGHTMAP;
    for (i = 0;(count >= widthset[i] break;
/* printf("index = %d \n", currentlineblock->index, image_row); */

    if ((((gap(current[inablock->current_block, widthfill[x], headblock))){

        &&(current[inablock->right_x-current[inablock->left_x > text_len*10)

        currentlineblock->side_gap |= RIGHTMAP;
        break;
    }
    add_block_into_line(widthfill[x], headblock, currentlineblock);
    tmp_block = widthfill[x];
    widthfill[x] = NULL;
}
```

```
        currentlineblock->att = TEXT;
        currentlineblock->width = currentlineblock->right_x -
            currentlineblock->left_x + 1;
        currentlineblock->length = currentlineblock->lower_y -
                                   currentlineblock->upper_y + 1;
        width_fill[stopx] = NULL;
}

/*****************************************************************************/
/******** test if one line totally inside the other line             ********/
/*****************************************************************************/

int line_inside(first, second)
struct lineblock *first, *second;
{
    if ((first->left_x >= second->left_x)&&(first->right_x <= second->right_x)&&
        (first->upper_y >= second->upper_y)&&(first->lower_y <= second->lower_y))
       return 1;
    else
       return 0;
}

/*****************************************************************************/
/******** check if the line below the picture, hpicture or table     ********/
/*****************************************************************************/
struct block_rect *if_caption(line, block)
struct lineblock *line;
struct block_rect *block;
{
    struct block_rect *tmp;

    for ( tmp = block->firstincontextblock; tmp != NULL; tmp = tmp->next){
        if (tmp->upper_y >= line->upper_y) break;
        if ((tmp->att&(HPICTURE|PICTURE|TABLE)) == 0) continue;
        else if ((over_range(line->left_x, line->right_x, tmp->left_x,
                 tmp->right_x) == 1.0) &&(line->upper_y <= tmp->lower_y+text_len)&&
                 (line->upper_y>tmp->lower_y)) {
printf("--------in caption block->index =%d x1 = %d y1 = %d x2 = %d y2 = %d\n",
   tmp->index, tmp->left_x, tmp->upper_y, tmp->right_x, tmp->lower_y);
printf("line->index =%d x1 = %d y1 = %d x2 = %d y2 = %d\n ",
   line->index, line->left_x, line->upper_y, line->right_x, line->lower_y);
                 line->att |= CAPTION;
                 line->under_caption = tmp;
                 line->l_caption = tmp->left_x;
                 line->r_caption = tmp->right_x;
                 return tmp;
               }
    }
    return NULL;
}

/*****************************************************************************/
/******** test if gapline between the lineblock                      ********/
/*****************************************************************************/
int if_gapline(first, second)
struct lineblock *first, *second;
{
       int center_x,  count1, i;
       int bound1, bound2;

       bound1 = min(first->upper_y, second->upper_y);
       bound2 = max(first->lower_y, second->lower_y);

       for (center_x = first->right_x+1; center_x < second->left_x; center_x++){  /*
           for ( i = bound1, count1 = 0; i <= bound2; i++)
               if (original_p[i][center_x]&&PHAMK) count1++;
           if (count1 >= (bound2-bound1+1)) return 1;
       }
       return 0;
}

/*****************************************************************************/
/******** test if vertical position in the same level                ********/
/*****************************************************************************/

int same_line(firstline, secondline, head)
struct lineblock *firstline, *secondline;
struct block_rect *head;
{
    struct block_rect *tmp1, *tmp2;
    struct lineblock *tmpline;
    int    if_neighbor;
    int    x1, y1, x2, y2, i, j;


    tmp1 = if_caption(firstline, head);
/* printf("after if_caption\n"); */
    tmp2 = if_caption(secondline, head);
/* printf("after if_caption\n"); */
  if (((tmp1 != NULL)&&(tmp2 == NULL))||((tmp1 == NULL)&&(tmp2 != NULL))||
      ((tmp1 != NULL)&&(tmp2 != NULL)&&(tmp1 != tmp2)))
     return 0;

  if (firstline->left_x > secondline->left_x){
     if (!if_gapline(secondline, firstline)) return 0;
     x1 = secondline->right_x+1;
     y1 = min(firstline->upper_y, secondline->upper_y);
     x2 = firstline->left_x-1;
     y2 = max(firstline->lower_y, secondline->lower_y);
  } else {
          if (!if_gapline(firstline, secondline)) return 0;

          x1 = firstline->right_x+1;
          y1 = min(firstline->upper_y, secondline->upper_y);
          x2 = secondline->left_x-1;
          y2 = max(firstline->lower_y, secondline->lower_y);

        }
  for ( i = x1; i <= x2; i++)
      for (j = y1; j <= y2; j++)
          if (original_p[j][i]&0x01) return 0;
  for (if_neighbor = 0, tmpline = secondline->previous; tmpline != NULL,
                                                  tmpline = tmpline->previous){
     if (tmpline->upper_y < min(firstline->upper_y, secondline->upper_y)
                                    -(int)(4.0*text_len))
        break;
     else if (if_overlap(tmpline->left_x, tmpline->right_x,
                  firstline->left_x,  firstline->right_x)&&
              if_overlap(tmpline->left_x, tmpline->right_x, secondline->left_x,
                  secondline->right_x)&&
              (tmpline->lower_y > min(firstline->upper_y, secondline->upper_y)
                  -(int)(2.0*text_len))){
           if_neighbor = 1;
           break;
        }
  }
  if (!if_neighbor == 0){
     for (tmpline = secondline->next; tmpline != NULL,
                                                  tmpline = tmpline->next){
        if ((!if_overlap(tmpline->left_x, tmpline->right_x, x1, x2))&&
            (!if_overlap(tmpline->upper_y, tmpline->lower_y, y1, y2)))
```


[図91]

【図92】

```
    return 0;
}

    if (tmpline->upper_y > max(firstline->lower_y, secondline->lower_y)
            +(int)(2.0*ext_ml))
        break;
    else if (!(_overlap(tmpline->left_x, tmpline->right_x,
            firstline->left_x, firstline->right_x)&&
        !(_overlap(tmpline->left_x, tmpline->right_x, secondline->left_x,
            secondline->right_x))){
        if_neighbor = 1;
        break;
    }
}

if (!if_neighbor == 1){
printf("blank_y=%d, %d %d-%d y=%d, %d %d=%d x1-%d x2-%d y1-%d y2-%d ) = %d\n",
        firstline->upper_y, firstline->lower_y,
        secondline->upper_y, secondline->lower_y,
        tmpline->upper_y, tmpline->lower_y,
        tmpline->left_x, tmpline->right_x, if_neighbor);
}

    if (!if_neighbor == 1){
    if (!(over_range(firstline->upper_y,
            secondline->upper_y, secondline->lower_y) >= LITTOOVERLAP)||
        (over_range(secondline->upper_y, secondline->lower_y,
            firstline->upper_y, firstline->lower_y ) >= LITTOOVERLAP)){
/*      (min(firstline->width, secondline->width) <= text_len*3)) */
printf("firstline = %d secondline = %d tmpline = %d \n", firstline->index,
    secondline->index, tmpline->index,
    return 1;
    }
}

    return 0;
}

/*****************************************************************************/
/*                                                                           */
/*********  sort line                                                        */
/*                                                                           */
/*****************************************************************************/
void sort_lines(firstlineblock)
struct lineblock *firstlineblock;
{
    struct lineblock *tmp;
    struct lineblock --line_x;
    int   num, i, j, step;

    if (firstlineblock->firstenbline == NULL) return;
    for (num = 0, tmp = firstlineblock->firstenbline; tmp != NULL; tmp = tmp->next)
        num++;

    if (num > 0) {
        line_x = new_linepointer(0, num-1);
        for (i = 0, tmp = firstlineblock->firstenbline; tmp != NULL; tmp = tmp->next)
            line_x[i++] = tmp;
    }

    /* with testing */

    sort_linepointer(line_x, num);

    firstlineblock->firstenbline = line_x[0];
    firstlineblock->lastenbline = line_x[num-1];
    if (num > 1){
        line_x[0]->next = line_x[1];
        line_x[0]->previous = NULL;
        line_x[num-1]->next = NULL;
        line_x[num-1]->previous = line_x[num-2];
        line_x[0]->index = num;
    }
    for ( i = 1; i < num-1; i++){
        line_x[i]->previous = line_x[i-1];
        line_x[i]->next = line_x[i+1];
        line_x[i]->index = i+1;
    }
    free((struct lineblock **)line_x);
}

/*****************************************************************************/
/*                                                                           */
/*********  combine full overlapping lines                                   */
/*                                                                           */
/*****************************************************************************/
void combine_full_overlap_line(first, firstblock)
struct lineblock *first;
struct block_rect *firstblock;
{
    struct linebloch *tmpline, *subtmpline, *tmp;
    int i;
    float gapmean;

    if (first->firstsubline == NULL) return;
    if (gap_nonover != 0) {
        gapmean = ((float)total_gap/(float)gap_nonover;
printf("widthsd = %d total_gap = %d gap_nonover = %d gap_overwid mean = %f\n",
            widthsd, total_gap, gap_over, gapmean);

        widthsd = (int)((float)(gap_over-widthsd)/
                ((float)widthsd-gapmean)*(float)(gap_nonover));
    }
printf("widthsd = %d total_gap = %d gap_nonover = %d gap_over=%d image_rowsize\n",
        widthsd, total_gap, gap_nonover, gap_over, image_rowsize);

    sort_line(first);
    for (tmpline=first; tmpline != NULL; tmpline = tmpline->next)
        for (subtmpline = tmpline->previous; subtmpline != NULL; ){
            if (!if_overlap(subtmpline->upper_y, subtmpline->lower_y,
                    tmpline->upper_y, tmpline->lower_y)){
                if (!(_overlap(subtmpline->left_x, subtmpline->right_x,
                        tmpline->left_x, tmpline->right_x)) {

                    if (line_inside(tmpline, subtmpline)){
                        line_inside(subtmpline, tmpline);
                        tmp = subtmpline->previous;
                        line_transfer(subtmpline, first, tmpline, TEXT);
                        subtmpline = tmp;
                        continue;
                    }
                }
                if (same_line(subtmpline, tmpline, firstblock)){
                    tmp = subtmpline->previous;
                    line_transfer(subtmpline, first, tmpline, TEXT);
                    subtmpline = tmp;
                    continue;
                }
            }
            else {
```

```
                    if (((subtmpline->right_x < tmpline->left_x)&&
                         (subtmpline->right_x >= tmpline->left_x-widthstd)&&
                         (!((tmpline->side_gap&LEFTGAP))&&
                          (!(subtmpline->side_gap&RIGHTGAP))))||
                        ((tmpline->right_x < subtmpline->left_x)&&
                         (tmpline->right_x >= subtmpline->left_x-widthstd)&&
                         (!((tmpline->side_gap&RIGHTGAP))&&
                          (!(subtmpline->side_gap&LEFTGAP))))){
printf("index 1 = %d index 2 = %d\n", subtmpline->index, tmpline->index);
                        if (same_line(subtmpline, tmpline, firstblock)){
printf("** index 1 = %d index 2 = %d\n", subtmpline->index, tmpline->index);
                            tmp = subtmpline->previous;
                            line_transfer(subtmpline, first, tmpline, TEXT);
                            subtmpline = tmp;
                            continue;
                        }
                    }
                }
*/
                subtmpline = subtmpline->previous;
            }
        for (tmpline=first->firstsubline, i = 1; tmpline != NULL;
                                            tmpline = tmpline->next, i++)
            tmpline->index = i;
}


/*****************************************************************************/
/******** Formation of line block                                    ********/
/*****************************************************************************/

void line_block(headlineblock, firstblock, column)
struct linesblock  *headlineblock;
struct block_rect *firstblock;
int    column;
{

    int    i, j, stopx, sort_index, num;
    int    numlineblock;
    struct block_rect **widthfill;
    struct block_rect *current_block;
    struct block_rect **sorted_block;
    struct lineblock  *tmplineblock;
    fillcondition     fill;


    total_gap = 0; gap_over = 0; gap_nonover = 0;
    upper_row = firstblock->upper_y;
    lower_row = firstblock->lower_y;
    upper_row = (upper_row-1)*reduce_factor;
    lower_row = lower_row*reduce_factor-1;
    if (lower_row >= image_row) lower_row = image_row;

    numlineblock = 0;
    text_len = (firstblock->avg_height != 0.0)?
                                    firstblock->avg_height:text_length;

    widthstd = (int)((float)text_len*WIDTHSTD+0.5);

printf("len = %f std = %d \n", text_len, widthstd);
    widthfill = new_blockpointer(0, column-1);
    for ( i = 0; i < column; widthfill[i++] = NULL);
    sorted_block = sort_textblock(&num, firstblock);
    text_area = plot_area(num, sorted_block, firstblock->upper_y,
                          firstblock->lower_y, firstblock->left_x,
                          firstblock->right_x);

    for (sort_index = 0; sort_index < num; ){
        fill=fill_test(&stopx, sorted_block[sort_index], firstblock, widthfill);

      /* group a new line */

      if (fill == NEXT){
         tmplineblock = new_line(headlineblock);
         tmplineblock->index = ++numlineblock;
         group_connected_block(stopx, column, tmplineblock, firstblock,
                                                          widthfill);
         continue;
      } else if (fill == SURROUNDIN) {
               converto_headblock(widthfill[stopx]);
               transfer_block(TEXT, TEXT, sorted_block[sort_index],
                              firstblock, widthfill[stopx]);
            } else {
                    for ( j = sorted_block[sort_index]->left_x;
                                j <= sorted_block[sort_index]->right_x; j++)
                        widthfill[j] = sorted_block[sort_index];
                }
        sort_index++;
    }

    for (  i = 0; i < column; i++){
        if (widthfill[i] != NULL){
            tmplineblock = new_line(headlineblock);
            stopx = i;
            tmplineblock->index = ++numlineblock;
            group_connected_block(stopx, column, tmplineblock,firstblock,
                                                          widthfill);
            i = tmplineblock->right_x;
        }
    }
  combine_full_overlap_line(headlineblock, firstblock);
printf("************ after first combine ************\n");
  combine_full_overlap_line(headlineblock, firstblock);

  free_cmatrix(text_area, firstblock->upper_y, firstblock->lower_y,
                          firstblock->left_x, firstblock->right_x);
  free((struct block_rect**)widthfill);
  free((struct block_rect**)sorted_block);
}


/*****************************************************************************/
/******** group line for the block contain more that one group area  ********/
/*****************************************************************************/

void line_group(firstlineblock, firstblock)
struct lineblock *firstlineblock;
struct block_rect *firstblock;
{
    int i, x, y, j, center_y;
    struct lineblock **tmpline;
    struct block_rect *tmpblock, *nextblock;
    struct whiteblock *tmpwhite;

    tmpline = new_linepointer(1, firstblock->group);
    for ( i = 1; i <= firstblock->group; i++)
        tmpline[i] = NEW_line(firstlineblock);

    for (tmpblock = firstblock->firstsubblock; tmpblock != NULL; ){
        nextblock = tmpblock->next;
        fadd_block_into_line(tmpblock, firstblock, tmpline[tmpblock->in_group]);
        tmpblock = NextBlock;
    }
```

【図94】

```
for (tmpwhite = firstblock->firstwhite; tmpwhite != NULL; tmpwhite = tmpwhite->next)
printf("white index = %d x1 = %d y1 = %d x2 = %d y2 = %d \n",
                tmpwhite->index, tmpwhite->left_x, tmpwhite->upper_y,
                tmpwhite->right_x, tmpwhite->lower_y);

for (tmpwhite = firstblock->firstwhite, i = 0; tmpwhite != NULL;
                i++, tmpwhite = tmpwhite->next)
        {

    if (tmpline[i]->first_block == NULL){
        tmpline[i]->left_x = (tmpwhite->right_x+tmpwhite->left_x)/2;
        tmpline[i]->right_x = (tmpwhite->right_x+tmpwhite->left_x)/2;
        tmpline[i]->upper_y = tmpwhite->upper_y-1;
        tmpline[i]->lower_y = tmpwhite->lower_y-1;
    }

if ((tmpline[i]->lower_y<tmpline[i]->upper_y+1) < MPOINT/SCALE_RATIO)4
        {tmpline[i]->first_block != NULL)){
    for (y = tmpline[i]->upper_y-1; j < 15 <= STANDARDFONT/2)4(y > 0);
                                                y--, j++){
        for (x = tmpline[i]->right_x; x <= tmpline[i]->left_x; x--)
            if (original_p[y][x]&0x01)  break;
        if (original_p[y][x]&0x01)  break;
    }
    if (tmpline[i]->upper_y-1 <= y) tmpline[i]->lower_y = y-2;
}

for (x = tmpline[i]->left_x; x > tmpwhite->left_x-1; x--)
    for (y = tmpline[i]->upper_y-1; y <= tmpline[i]->lower_y-2; y++)
        if (original_p[y][x]&0x01){
                tmpline[i]->left_x = x;
                break;
            }

for (x = tmpline[i]->right_x; x < tmpwhite->right_x-1; x++)
    for (y = tmpline[i]->upper_y-2; y <= tmpline[i]->lower_y-2; y++)
        if (original_p[y][x]&0x01){
                tmpline[i]->right_x = x;
                break;
            }

tmpline[i]->length = tmpline[i]->lower_y-tmpline[i]->upper_y+1;
tmpline[i]->width = tmpline[i]->right_x-tmpline[i]->left_x+1;
}

firstlineblock->firstwhite = tmpline[2];
firstlineblock->currentmline = tmpline[firstblock->group];
for (i = 3; i <= firstblock->group; i++){
    tmpline[i]->next = NULL;
    tmpline[i]->index = i;
}

tmpline[i]->next = tmpline[2];
tmpline[firstblock->group]->previous = tmpline[firstblock->group-1];
for (i = 2; i < firstblock->group; i++){
    tmpline[i]->next = tmpline[i+1];
    tmpline[i]->previous = tmpline[i-1];
}
free((struct linablock **)tmpline);
}

/*.....group line for the picture block containing more than one group area.....*/
/*.....*/

void line_picsum(firstlineblock, firstblock)
struct linablock *firstlineblock;
struct block_rect *firstblock;
{
    int i, j, line;
    struct linablock **tmpline;
    struct block_rect *tmpblock, *nextblock;

tmpline = new_linapointer(i, firstblock->group);
for ( i = 2; i <= firstblock->group; i--)
    tmpline[i] = new_linblock(0,0);

for (tmpblock = firstblock->firstnubblock; tmpblock != NULL; ){
    nextblock = tmpblock->next;
    read_block_into_line(tmpblock, firstblock, tmpline[tmpblock->in_group]);
    tmpblock = nextblock;
}

for ( i = 1, line = 0; i <= firstblock->group; i++){
    if (tmpline[i]->first_block != NULL){
        firstlineblock->currentmline = tmpline[i];
        tmpline[i]->length = tmpline[i]->lower_y-tmpline[i]->upper_y+1;
        tmpline[i]->width = tmpline[i]->right_x-tmpline[i]->left_x+1;
        tmpline[i]->set = TRUE;
        tmpline[i]->index = line++;
    }
}

for ( j = i-1, i = 2; i <= firstblock->group; j--){
    if (tmpline[j]->first_block != NULL){
        tmpline[j]->previous = firstlineblock->currentmline;
        firstlineblock->currentmline = tmpline[j];
        tmpline[j]->next = NULL;
        tmpline[j]->length = tmpline[j]->lower_y-tmpline[j]->upper_y+1;
        tmpline[j]->width = tmpline[j]->right_x-tmpline[j]->left_x+1;
        tmpline[j]->set = TRUE;
        tmpline[j]->index = line++;
    } else free((struct linablock *)tmpline[j]);
    break;
}
} else free((struct linablock **)tmpline[1]);

free((struct linablock **)tmpline+1);
}

/*.....*/
/*..... make gap on the left and right side of picture .....*/
/*.....*/

void make_gap(firstblock)
struct block_rect *firstblock;
{
    struct block_rect *tmp;
    int   x, y, count;

for (tmp = firstblock->firstnextblock; tmp != NULL; tmp = tmp->next)
    if ((tmp->area[PICTURE_IMAGE][PICTURE_TABLE][WORD]) ||(
```

【図95】

```
printf("make_grp on %d x - %d y - %d \n", tmp->xindex, tmp->left_x, tmp->upper_y);
        x = tmp->left_x -1;
        count = 0;
        x = tmp->left_x-1;
/*      for ( x = tmp->left_x-1; x >= max((tmp->left_x-(int)rect_length),
                firstblock->left_x); x--){
*/
        for (y = tmp->upper_y; y <= tmp->lower_y; y++)
            if (original_p[y][x] != DARKAGE) count++;
printf("count - %d x1 - %d y1 - %d y2 - %d y1-%d \n", count, x, tmp->upper_y, tmp->low
er_y,
                firstblock->left_x, tmp->right_x, tmp->upper_y, tmp->lower_y);
        if (count <= NOISE_NO){
            for (y = tmp->upper_y; y <= tmp->lower_y, y++)
                original_p[y][x] = DARKAGE;
            for (y = tmp->lower_y-1; y <= firstblock->lower_y ; y++)
                if (original_p[y][x] == DARKAGE) break;
                else original_p[y][x] = DARKAGE;
        }
/*      }
*/
/*      x = tmp->right_x;
*/
/*      for (x = tmp->right_x +1; x <= min((tmp->right_x+(int)rect_length),
                firstblock->right_x); x++){
*/
        count = 0;
        for (y = tmp->upper_y; y <= tmp->lower_y; y++)
            if (original_p[y][x] == DARKAGE) count++;
printf("count - %d x1 - %d y1 - %d y2 - %d y1-%d \n", count, x, tmp->upper_y, tmp->low
er_y,
                firstblock->lower_y);
        if (count <= NOISE_NO){
            for (y = tmp->upper_y; y <= tmp->lower_y, y++)
                original_p[y][x] = DARKAGE;
            for (y = tmp->lower_y+1; y <= firstblock->lower_y ; y++)
                if (original_p[y][x] == DARKAGE) break;
                else original_p[y][x] = DARKAGE;
        }
/*      }
*/
    }
}

/*************************************************************************/
/*******  Initialize the formation of line block                 *********/
/*************************************************************************/

void ini_line_block(firstblock, firstblock, column, orig_row, o_p)
struct linablock *firstlineblock;
struct block_rect *firstblock;
int               column, orig_row
unsigned char     **o_p;
{
    int  num;
    struct block_rect *tmpb, *tmpb1;
    struct linablock  *tmpline;

printf("get into line \n");
    image_row = orig_row;
    original_p = o_p;
    firstlineblock->first_block = firstlineblock->current_block = firstblock;
    make_grp(firstblock);
    if (firstblock->group > 1)
        firstblock->group = firstblock->group;
    if (firstblock->attribute & (TABLE))
        line_group(firstlineblock, firstblock);
    else if ((firstblock->attribute&(FRAME|GRAPHIC|PICTURE)&&(firstblock->group >=2)&&
            (((firstblock->attribute&(FRAME)))
        line_picture(firstlineblock, firstblock);
    else  if (((firstblock->firstnumblock != NULL)||(
            line_block(firstlineblock, firstblock, column);
    }
    printf("get into line 1 \n");

    for (num=0,tmpb=firstblock->firstnontextblock; tmpb != NULL; ){
        tmpline = new_nontextline(firstlineblock);
        tmpline->right_x = tmpb->right_x;
        tmpline->left_x = tmpb->left_x;
        tmpline->upper_y = tmpb->upper_y;
        tmpline->lower_y = tmpb->lower_y;
        tmpline->width = tmpb->width;
        tmpline->length = tmpb->length;
        tmpline->indent = *num;
        tmpline->att = tmpb->att;
        tmpb1 = tmpb->next;
        tmpb->next=NULL;
        tmpb->previous = NULL;
        if(((tmpb->attribute&(PICTURE))&&((tmpb->attribute&(WORD))
            &&((tmpb->attribute&(TEXT))&&((firstblock->attribute&(FRAME))))
            in_line_block(tmpline, tmpb, column, orig_row, o_p);
        else {
            tmpline->first_block = tmpb;
            tmpline->current_block = tmpb;
        }
        tmpb = tmpb1;
    }
}

/*************************************************************************/
/*******  Print the line information into a file                 *********/
/*************************************************************************/

void print_ln(level, firstline)
int  level;
struct linablock *firstline;
{
    int   nestlevel, i;
    struct linablock *tmpenable, *tmpline;

    nestlevel = level;
    for (tmpenable = firstline->firstenabling; tmpenable != NULL;
                                        tmpenable = tmpenable->next){
        for ( i = 0; i <= (level-1)*6; i++)
            printf(line_file, " ");
        fprintf(line_file,"id index = %d ", tmpenable, tmpenable->index);
        printf(line_file, "next yoko left wide num_subblock = %d att=%x under_space%d\n",
                            tmpenable->left_x, tmpenable->upper_y,
                            tmpenable->length, tmpenable->width,
                            tmpenable->numsubblock, tmpenable->att,
                            tmpenable->under_space[on-blank]);
    }

    for (tmpline = firstline->firstnontextline; tmpline != NULL;
```

【図96】

```
                                                    tmpline=tmpline->next){
            for ( i = 0; i <= (level-1)*8; i++)
                fprintf(line_file, " ");
            fprintf(line_file, "%d nontext index = %d ", tmpline, tmpline->index);
            fprintf(line_file,
  "x=%d y=%d l=%d w=%d att = %x block index1=%d index2=%d r = %d l = %d r = %d\n",
                    tmpline->left_x, tmpline->upper_y, tmpline->length,
                    tmpline->width, tmpline->att, tmpline->first_block->index,
                tmpline->current_block->index, tmpline->group, tmpline->l_caption,
                tmpline->r_caption);
            print_ln(nextlevel, tmpline);
        }
}


/*************************************************************************/
/******** Open the file for the output of line message           ********/
/*************************************************************************/

void print_lineblock(firstline)
struct lineblock *firstline;
{
        line_file = fopen("line_file.dat", "w");
        print_ln(1, firstline);
        fclose(line_file);
}
```

【図118】

```
                                                    tmpsection = tmpsection->next){
        if ((tmpsection->att&(TITLE|HEADLID|FOOTLID|PICTURE))) continue;
        else
            ini_section_block(tmpsection, tmpsection->firstlineblock, p,
                                                            column);
    }
}


/*************************************************************************/
/******** Print the line information into a file                ********/
/*************************************************************************/

void print_sec(level, firstsec)
int    level;
struct sectionblock *firstsec;
{
        int    nextlevel, i;
        struct sectionblock *tmpsubsec,  *tmpsec;

        nextlevel = level+1;
        for (tmpsubsec = firstsec->firstsubsection; tmpsubsec != NULL;
                                                    tmpsubsec = tmpsubsec->next){
            for ( i = 0; i <= (level-1)*8; i++)
                printf(" ");
                printf("index = %d ", tmpsubsec->index);
                printf("x=%d y=%d l=%d w=%d att = %x \n",
                        tmpsubsec->left_x, tmpsubsec->upper_y,
                        tmpsubsec->length, tmpsubsec->width, tmpsubsec->att);

        }

        for (tmpsec = firstsec->firstnontextsection; tmpsec != NULL;
                                                    tmpsec=tmpsec->next){
            for ( i = 0; i <= (level-1)*8; i++)
                printf(" ");
            printf("nontext index = %d ", tmpsec->index);
            printf("x=%d y=%d l=%d w=%d att = %x \n",
                    tmpsec->left_x, tmpsec->upper_y, tmpsec->length,
                    tmpsec->width, tmpsec->att);
            print_sec(nextlevel, tmpsec);
        }
}


/*************************************************************************/
/******** Open the file for the output of line message          ********/
/*************************************************************************/

void print_secblock(firstsec)
struct sectionblock *firstsec;
{
      print_sec(1, firstsec);
}
```

【図137】

Source Code for

Proportional Spaced Segmentation

【図97】

```
/*****************************************************************/
/*                                                               */
/*                  File: blocksection.c                         */
/*                  Author: Shih-Yuan Wang                       */
/*                  Date : 1-21-93                               */
/*                                                               */
/*****************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include "blocksection.h"
#include "blockgeneral.h"

float  tatal;
extern float text_length;
unsigned char *original_y,
int    min_y, max_y;
struct segment_gap *segment;
int    current_seg = 0;

/*****************************************************************/
/*                                                               */
/*        absolute difference of two integers                    */
/*                                                               */
/*****************************************************************/

int abs(a, b)
int a, b;
{
    return (a>b)?a-b:b-a;
}

/*****************************************************************/
/*                                                               */
/*        allocation lineblockpointer                            */
/*                                                               */
/*****************************************************************/

struct lineblock **new_linepointer(nl, nh)
int nl, nh;
{
  struct lineblock **v;

  v = (struct lineblock **) malloc((nh-nl+1)*sizeof(struct lineblock*));
  if (!v) {
    printf("error: in the allocation of linepointer\n");
    exit(1);
  }
  v -= nl;
  return v;
}

/*****************************************************************/
/*                                                               */
/*        allocation new segment_gap structure                   */
/*                                                               */
/*****************************************************************/

struct segment_gap *new_segment(nl, nh)
int nl, nh;
{
  struct segment_gap **v;
  int   i;

  v = (struct segment_gap *) malloc((nh-nl+1)*sizeof(struct segment_gap));
  if (!v) {
    printf("error: in the allocation of segment_gap\n");
    exit(1);
  }
  v -= nl;

  for ( i = nl; i <= nh; i++)
    v[i].column_no = 0;
  return v;
}

/*****************************************************************/
/*                                                               */
/*        allocation sectionblockpointer                         */
/*                                                               */
/*****************************************************************/

struct sectionblock **new_sectionpointer(nl, nh)
int nl, nh;
{
  struct sectionblock **v;

  v = (struct sectionblock **) malloc((nh-nl+1)*sizeof(struct sectionblock*));
  if (!v) {
    printf("error: in the allocation of sectionpointer\n");
    exit(1);
  }
  v -= nl;

  return v;
}

/*****************************************************************/
/*                                                               */
/*        sort line block pointer array                          */
/*                                                               */
/*****************************************************************/

void sort_linepointer(line, num)
struct lineblock *line;
int   num;
{
  int   i, j, stop;
  struct lineblock *tmp;

  for (i = 0; i < num-1 ; i++){
    for (j = 0, stop = 0; j < num-1-i; j++)
      if (line[j]->upper_y > line[j+1]->upper_y){
        tmp = line[j+1];
        line[j+1] = line[j];
        line[j] = tmp;
        stop = 1;
      }
    if (stop == 0) break;
  }
}

/*****************************************************************/
/*                                                               */
/*        line block sorting                                     */
/*                                                               */
/*****************************************************************/

struct lineblock **sort_y(firstline, maxline)
struct lineblock *firstline;
int   maxline;
{
  struct lineblock  *tmp;
  struct lineblock  **line_y;
  int   i, j, stop;

  maxline = 0;
  for ( tmp = firstline; tmp != NULL; tmp = tmp->next)
    maxline++;
  tmp = firstline;
  for ( i = 0; i < maxline; i++, tmp = tmp->next)
    line_y = new_linepointer(0, maxline-1);
```

[図98]

```
    for (i = 0, tmp = firstlineblock->firstsubline; tmp != NULL; tmp = tmp->next)
        line_x[i++] = tmp;

    /* extra testing */

    if ((firstlineblock->currentsubline != NULL)&&
        (i != firstlineblock->currentsubline->index))
        printf("error: in sort_y\n");

    for (tmp = firstlineblock->firstnontextline; tmp != NULL; tmp = tmp->next)
        line_x[i++] = tmp;

    sort_linepointer(line_x, *numline);
    return line_x;
}

/*****************************************************************************/
/********  section block sorting                                    ********/
/*****************************************************************************/

struct sectionblock **sort_section(first, num)
struct sectionblock *first;
int     *num;
{
    struct sectionblock *tmp;
    struct sectionblock **line_x;
    int     i, j, stop;

    *num = 0;
    for (tmp = first->firstsubsection; tmp != NULL; tmp = tmp->next)
        (*num)++;

    line_x = new_sectionpointer(0, (*num)-1);

    for (i = 0, tmp = first->firstsubsection; tmp != NULL; tmp = tmp->next)
        line_x[i++] = tmp;

    for ( i = 0; i < (*num) -1; i++){
        for (j = 0, stop = 0; j < (*num)-1-i; j--)
            if ((line_x[j]->upper_y > line_x[j+1]->upper_y) ||
                ((line_x[j]->upper_y == line_x[j+1]->upper_y)&&
                 (line_x[j]->lower_y > line_x[j+1]->lower_y))) {
                tmp  = line_x[j-1];
                line_x[j-1] = line_x[j];
                line_x[j] = tmp;
                stop = 1;
            }
        if (stop == 0) break;
    }
    return line_x;
}

/*****************************************************************************/
/********  Allocate new line block                                  ********/
/*****************************************************************************/

struct sectionblock *new_secblock(nl, nh)
int nl, nh;
{
        int i;
        struct sectionblock *v;

        v = (struct sectionblock *) malloc((unsigned)(nh-nl+1)*
                                           sizeof(struct sectionblock));
        if (!v){
           printf("allocation error: section block \n");
           exit(1);
        }
        v = v-nl;
        for ( i = nl; i <=nh; i--) {
            v[i].firstlineblock = v[i].currentlineblock = NULL;
            v[i].firstsubsection = v[i].currentsubsection = NULL;
            v[i].firstnontextsection = v[i].currentnontextsection = NULL;
            v[i].previous = v[i].next = NULL;
            v[i].upper_y = v[i].lower_y = v[i].left_x = v[i].right_x = 0;
            v[i].width = v[i].length = 0;
            v[i].att = UNKNOWN;
            v[i].numlineblock = 0;
            v[i].index = 0;
            v[i].boundary_index = -1;
        }
        return v;
}

/*****************************************************************************/
/********  Chain the nontext section block                          ********/
/*****************************************************************************/

struct sectionblock *new_ntsection(headsection)
struct sectionblock *headsection;
{
        struct sectionblock *n;

        n = new_secblock(0,0);
        if (headsection->firstnontextsection == NULL)
           headsection->firstnontextsection = n;
        else{
            headsection->currentnontextsection->next = n;
            n->previous = headsection->currentnontextsection;
        }
        headsection->currentnontextsection = n;
        return n;
}


/*****************************************************************************/
/********  Chain the section block                                  ********/
/*****************************************************************************/

struct sectionblock *new_section(headsection)
struct sectionblock *headsection;
{
        struct sectionblock *n;

        n = new_secblock(0,0);
        if (headsection->firstsubsection == NULL)
           headsection->firstsubsection = n;
        else{
            headsection->currentsubsection->next = n;
            n->previous = headsection->currentsubsection;
        }
        headsection->currentsubsection = n;
        return n;
}


/*****************************************************************************/
/********  take lineblock from section                              ********/
/*****************************************************************************/

void take_section(line,  currsection)
struct lineblock    *line;
struct sectionblock *currsection;
```

【図99】

```
{
    if (currection->firstlineblock == line){
        currection->firstlineblock = line->next;
        if (currection->currentlineblock == line)
            currection->currentlineblock = NULL;
        else
            currection->firstlineblock->previous = NULL;
    } else if (currection->currentlineblock == line){
        currection->currentlineblock = line->previous;
        line->previous->next = NULL;
    } else {
        line->previous->next = line->next;
        line->next->previous = line->previous;
    }
}
/*.....put lineblock into section.....*/
/*.....*/

void put_section(line, currection)
struct lineblock    *line;
struct sectionblock *currection;
{
    if (currection->firstlineblock == NULL){
        currection->firstlineblock = line;
        currection->currentlineblock = line;
        line->next = NULL;
        line->previous = NULL;
    } else {
        currection->currentlineblock->next = line;
        line->previous = currection->currentlineblock;
        currection->currentlineblock = line;
        line->next = NULL;
    }
}
/*.....create new sameline structure.....*/
/*.....*/

struct same_line *new_sameline(curr)
struct lineblock *curr;
{
    struct sameline *v;

    v = (struct sameline *) malloc(sizeof(struct sameline));
    if (!v) {
        printf("allocation error: in sameline\n");
        exit(1);
    }
    v->current = curr;
    v->previous = NULL;
    v->next = NULL;
    v->r_connection = v->l_connection = NULL;
    return v;
}
/*.....sort sameline array by x.....*/
/*.....*/

void sort_sameline(first)
struct sameline *first;
{
    int    stop, tapindex;
    struct sameline *tap;
    struct lineblock *tapline;

    for ( ; ; ){
        stop = 0;
        for (tap = first->next; tap != NULL; tap = tap->next)
            if (tap->current->line_x < tap->previous->current->left_x){
                tapline = tap->current;
                tapindex = tap->index;
                tap->current = tap->previous->current;
                tap->index = tap->previous->index;
                tap->previous->current = tapline;
                tap->previous->index = tapindex;
                stop = 1;
            }
        if (stop == 0) break;
    }
}
/*.....remove lineblock.....*/
/*.....*/

void line_rm(text, headline, line_rm)
int    text;
struct lineblock *headline, *line_rm;
{
    if (text == TEXT){
        if (headline->firstmline == line_rm){
            headline->firstmline = line_rm->next;
            if (line_rm->next != NULL) line_rm->next->previous = NULL;
            else headline->currentmline = NULL;
        } else if (headline->currentmline == line_rm){
            headline->currentmline = line_rm->previous;
            line_rm->previous->next = NULL;
        } else {
            line_rm->next->previous = line_rm->previous;
            line_rm->previous->next = line_rm->next;
        }
    } else {
        if (headline->firstcomline == line_rm){
            headline->firstcomline = line_rm->next;
            if (line_rm->next != NULL) line_rm->next->previous = NULL;
            else headline->currentcomline = NULL;
        } else if (headline->currentcomline == line_rm){
            headline->currentcomline = line_rm->previous;
            line_rm->previous->next = NULL;
        } else {
            line_rm->next->previous = line_rm->previous;
            line_rm->previous->next = line_rm->next;
        }
    }
}
/*.....combination of line block  forward.....*/
/*.....*/

void line_combine(newhead, line_rma)
struct lineblock *newhead, *line_rma;
{
```

```
    newhead->first_block->previous = line_tran->current_block;
    line_tran->current_block->next = newhead->first_block;
    newhead->first_block = line_tran->first_block;
    newhead->numsubblock += line_tran->numsubblock;
    if (line_tran->left_x < newhead->left_x)
                        newhead->left_x = line_tran->left_x;
    if (line_tran->right_x > newhead->right_x)
                        newhead->right_x  = line_tran->right_x;
    if (line_tran->upper_y < newhead->upper_y)
                        newhead->upper_y = line_tran->upper_y;
    if (line_tran->lower_y > newhead->lower_y)
                        newhead->lower_y = line_tran->lower_y;
    newhead->width = newhead->right_x-newhead->left_x+1;
    newhead->length = newhead->lower_y-newhead->upper_y+1;
}

/*****************************************************************/
/********  combination of line block backward            ********/
/*****************************************************************/

void bline_combine(newhead, line_tran)
struct lineblock *newhead, *line_tran;
{

    newhead->current_block->next = line_tran->first_block;
    line_tran->first_block->previous = newhead->current_block;
    newhead->current_block = line_tran->current_block;
    newhead->numsubblock += line_tran->numsubblock;
    if (line_tran->left_x < newhead->left_x)
                        newhead->left_x = line_tran->left_x;
    if (line_tran->right_x > newhead->right_x)
                        newhead->right_x  = line_tran->right_x;
    if (line_tran->upper_y < newhead->upper_y)
                        newhead->upper_y = line_tran->upper_y;
    if (line_tran->lower_y > newhead->lower_y)
                        newhead->lower_y = line_tran->lower_y;
    newhead->width = newhead->right_x-newhead->left_x+1;
    newhead->length = newhead->lower_y-newhead->upper_y+1;
}

/*****************************************************************/
/********  line remove from some head and merge with another head ********/
/*****************************************************************/

void line_transfer(line_tran, originalhead, newhead, att)
struct lineblock *line_tran, *originalhead, *newhead;
int    att;
{

    line_rm(att, originalhead, line_tran);
    bline_combine(newhead, line_tran);
    free((struct lineblock*) line_tran);
}

/*****************************************************************/
/********  test if one line lie inside another line      ********/
/*****************************************************************/

int if_inside(firstline, secondline)
struct linablock *firstline, *secondline;
{
    if (((float)cover_range(firstline->upper_y, firstline->lower_y,
                secondline->upper_y, secondline->lower_y) >= 1.0) &&
       ((float)cover_range(firstline->left_x, firstline->right_x,
                secondline->left_x,  secondline->right_x) >= 1.0))
      return 1;
    return 0;
}

/*****************************************************************/
/********  cover part                                    ********/
/*****************************************************************/

float cover_part(mark, a1, a2)
int *mark;
int a1, a2;
{
    int unmark, i;

    for ( i = a1, unmark = 0; i <= a2; i++){
        if (mark[i] == 0){
          unmark++;
          mark[i] = 1;
        }
    }
    return ((float)unmark/(float)(a2-a1+1));
}

/*****************************************************************/
/********  the number of "neighbor" on the left side     ********/
/*****************************************************************/

void text_left_neighbor(line, leftlevel, leftlength, head, linelist, nodistroy)
struct samaline *line;
int    *leftlevel;
float  *leftlength;
struct lineblock *head;
struct lineblock **linelist;
struct samaline *nodistroy;
{
    int    m, upper, lower, *mark, mostleft;
    struct samaline *maxline, *minline, *tmp;

    upper = line->current->upper_y;
    lower = line->current->lower_y;
    for (tmp = line->previous; tmp != NULL; tmp = tmp->previous){
        if (tmp->current == NULL) continue;
        if (tmp->current->upper_y < upper) upper = tmp->current->upper_y;
        if (tmp->current->lower_y > lower) lower = tmp->current->lower_y;
    }

    mark = (int *)ivector(upper, lower);
    mostleft = line->current->left_x;
    maxline = minline = NULL;
    *leftlevel = 0;
    *leftlength = 0.0;
    for ( tmp = line->previous; tmp != NULL; tmp=tmp->previous){
        if (tmp->current == NULL) continue;
        if ((tmp->current->right_x>mostleft-line->current->length)&&
            (cover_range(tmp->current->upper_y, tmp->current->lower_y,
             line->current->upper_y, line->current->lower_y)>0.0)) {
          if ((m = cover_part(mark,tmp->current->upper_y,tmp->current->lower_y)
                                                          > OVERLAP)
            {
              if (tmp->current->left_x < mostleft)
                              mostleft = tmp->current->left_x;
              *leftlength += ((float)tmp->current->length*m)
              if (m != 0.0) {
                (*leftlevel)++;
                if ((maxline == NULL)&&(minline == NULL)){
                  maxline = tmp;
                  minline = tmp;
```

【図101】

```
                } else if (tmp->current->length >= maxline->current->length)
                        maxline = tmp;
                    else
                      if (tmp->current->length < minline->current->length)
                        minline = tmp;
            }
        }
    }
}

if (*leftlevel == 2){
    if (if_overlap(minline->current->left_x, minline->current->right_x,
                   maxline->current->left_x, maxline->current->right_x)){
        if ((minline->current->width < minline->current->length*4)&&
            (*leftlength < ((float)line->current->length*1.5))&&
            ((maxline->current->length >= minline->current->length*3)||
             (maxline->current->width < maxline->current->length * 2))){

            *leftlevel = 1;

            if (nodistroy != minline){
                line_transfer(minline->current,head,maxline->current,
                                                minline->current->att);
                minline->current = NULL;
                linelist[minline->index] = NULL;
                minline = NULL;
            }
              else  {
                    line_transfer(maxline->current,head,minline->current,
                                                maxline->current->att);
                    maxline->current = NULL;
                    linelist[maxline->index] = NULL;
                    maxline = NULL;
                }
        }
    } else  {
            *leftlevel = 1;
            if (minline->current->left_x < maxline->current->left_x) {
                *leftlength = maxline->current->length;
                minline  = NULL;
            } else {
                    *leftlength = minline->current->length;
                    maxline = NULL;
                }
        }
    }
    if (*leftlevel == 1)
        line->l_connection = (maxline==NULL)?minline:maxline;


    if((*leftlevel >= 2) *leftlength /= ((float)*leftlevel;
    free((int *) (mark+upper));
}

/****************************************************************************/
/********  the number of "neighbor" on the right side                ********/
/****************************************************************************/

void text_right_neighbor(line, rightlevel, rightlength, head, linelist, nodistroy)
struct sameline *line;
int     *rightlevel;
float   *rightlength;
struct lineblock  *head, **linelist;
struct sameline *nodistroy;
{
    int     upper, lower, *mark, mostright;
    struct sameline *maxline, *minline, *tmp;
    float   m;


    upper = line->current->upper_y;
    lower = line->current->lower_y;
    for (tmp = line->next; tmp != NULL; tmp = tmp->next){
        if (tmp->current == NULL) continue;
        if (tmp->current->upper_y < upper) upper = tmp->current->upper_y;
        if (tmp->current->lower_y > lower) lower = tmp->current->lower_y;
    }

    mark = (int *)ivector(upper, lower);
    mostright = line->current->right_x;
    maxline = minline = NULL;
    *rightlevel = 0;
    *rightlength = 0.0;
    for ( tmp = line->next; tmp != NULL; tmp=tmp->next){
        if (tmp->current == NULL) continue;
        if (line_inside(tmp->current, line->current)){
            line_transfer(tmp->current, head, line->current, tmp->current->att);
            tmp->current = NULL;
            linelist[tmp->index] = NULL;
            continue;
        }
        if ((tmp->current->left_x<=mostright+line->current->length)&&
            (cover_range(tmp->current->upper_y, tmp->current->lower_y,
             line->current->upper_y, line->current->lower_y) > 0.0)) {
            m = cover_mark(mark,tmp->current->upper_y,tmp->current->lower_y);
printf("m = %f \n", m);
            if ((tmp->current->right_x <= mostright+1)||(m >= OVERLAP))
            {
                if (tmp->current->right_x > mostright)
                    mostright = tmp->current->right_x;
                *rightlength += (float)tmp->current->length*m;
                if (m != 0.0) {
                    (*rightlevel)++;

                    if ((maxline == NULL)&&(minline == NULL)){
                        maxline = tmp;
                        minline = tmp;
                    } else if (tmp->current->length >= maxline->current->length)
                            maxline = tmp;
                        else
                          if (tmp->current->length < minline->current->length)
                            minline = tmp;
                }
            }
        }
    }
  printf("right level in right_text %d \n", *rightlevel);
        if (*rightlevel == 2){
            if (if_overlap(minline->current->left_x, minline->current->right_x,
                           maxline->current->left_x, maxline->current->right_x)){
                if ((minline->current->width < minline->current->length*4)&&
                    (*rightlength < ((float)line->current->length*1.5))&&
                    ((maxline->current->length >= minline->current->length*3)||
                    (maxline->current->width < maxline->current->length * 2)))
                {
                    *rightlevel = 1;
                    if (nodistroy != minline){
                        line_transfer(minline->current,head,maxline->current,
                                                    minline->current->att);
                        minline->current = NULL;
                        linelist[minline->index] = NULL;
                        minline = NULL;
```

【図１０２】

```
                                                    text_right_neighbor(tmpline, rightlevel, rightlength, first, line, curr
                                    nextline);
                                }
                                else {
                                    line_transfer(mainline->current,head,mainline->current,
                                        mainline->current->size);
                                    mainline->current->left_x = mainline->current->left_x) {
                                        rightlength = mainline->current->length;
                                        nextline = NULL;
                                    } else {
                                        rightlength = mainline->current->length;
                                        mainline = NULL;
                                    }
                                }
                            } else {
                                rightlevel = 1;
                                if (mainline->current->left_x < mainline->current->left_x) {
                                    rightlength = mainline->current->length;
                                    nextline = NULL;
                                } else {
                                    rightlength = mainline->current->length;
                                    mainline = NULL;
                                }
                            }
                        }
                        if (*rightlevel == 1)
                            line->x_connection = (mainline==NULL)?nullline:mainline;
                        if(rightlevel >= 2) *rightlength /= (float)*rightlevel;
                        free((int *) mark-upper));
                    }

/*-------------------------------------------------------------------------*/
/*-------- text neighbor of some unknown block                            */
/*-------------------------------------------------------------------------*/

void text_neighbor(line,leftlevel,rightlevel,leftlen,rightlen,first,linelist, nodist)
struct mameline *line;
int   *leftlevel, *rightlevel;
float *leftlen, *rightlen;
struct lineblock *first;
struct lineblock **linelist;
struct mameline **nodist;
{
    text_left_neighbor(line, leftlevel, leftlen, first, linelist, nodist);
    text_right_neighbor(line, rightlevel, rightlen, first, linelist, nodist);
}

/*-------------------------------------------------------------------------*/
/*-------- Make title toward left direction                               */
/*-------------------------------------------------------------------------*/

void left_title(currentline, line, first)
struct mameline *currentline;
struct mameline *line;
struct lineblock **first;
{
    struct mameline *tmpline;
    int   leftlevel, rightlevel;
    float leftlength, rightlength;

    leftnest = currentline->left_x;
    for ( tmpline = currentline->x_connection; tmpline != NULL;
          tmpline = currentline->x_connection) {
        if (tmpline->current == NULL) continue;
printf("left = %d \n", tmpline->current->index);
        if (tmpline->current->right_x >= leftnest-0.5){
            text_left_neighbor(tmpline, &leftlevel, &leftlength, first, line, curr
                                                    text_left_neighbor(tmpline, &leftlevel, &leftlength, first, line, curr
            if (leftlevel == 1&&(tmpline->x_connection == currentline)){
                if (cover_ridge(tmpline->current->upper_y,
                    tmpline->current->lower_y, currentline->current->upper_y,
                    currentline->current->lower_y) >= 0.5) {
```

【図103】

```
    if (tmpline->current->right_x > rightmost)
        rightmost = tmpline->current->right_x;
    line_transfer(tmpline->current, first, currentline->current,
                  tmpline->current->attr);
    currentline->current->att = TITLE|TEXT;
    tmpline->current = NULL;
    line[tmpline->index] = NULL;
    set_right_neighber(currentline, rightlevel, rightlength,
                       first, line, NULL);
    if (rightlevel == 1)
        continue;
    else
        break;
    } else {
        tmpline->l_connection = NULL;
        currentline->current->att = TEXT;
        break;
    } else {
        tmpline->l_connection = NULL;
        break;
    }
} else break;
}
}

/*.....pack a set of line block.....*/
/*.....                         .....*/

void pack_lines(line, num)
struct linkblock **line;
int
{
    int
    i, j, pack;
    struct linkblock **tmpline;

    tmpline = new_linepointer(0, (*num)-1);
    pack = 0; j = 0, i < *num; i++){
    for ( ; line[i] == NULL;
        pack = 1;
    else
        tmpline[j++] = line[i];
}
if (pack == 1) {
    for ( i = 0; i < j; i++) line[i] = tmpline[i];
    for ( i = j; i < *num; i++) line[i] = NULL;
    *num = j;
}
free((struct linepointer **) tmpline);
}

/*.....                         .....*/
/*.....index for combining two samaine block.....*/
/*.....                         .....*/

int index_of_combine(line1, line2)
struct linkblock *line1, *line2;
{
    float i, a, p, q;

    i = cover_range(line1->upper_y, line1->lower_y, line2->upper_y,
                    line2->lower_y);

    a = cover_range(line1->left_x, line1->right_x, line2->left_x,
                    line2->right_x);

    p = cover_range(line2->left_x, line2->right_x, line1->left_x,
                    line1->right_x);

    q = cover_range(line2->left_x, line2->right_x, line1->left_x,
                    line1->right_x);

    if (((line1->att != TEXT)&&(line2->att != TEXT)) {
        if ((i >= 0.5)||(a >= 0.5)) return 1;
        else return 0;
    }
    if (((line1->att == TEXT)&&(line2->att != TEXT))){
        if (a >= 0.0) return 0;
        if (a < 1.0/3.0) return 2;
        else return 1;
    }
    if (((line1->att != TEXT)&&(line2->att == TEXT))){
        if (i >= 0.0) return 0;
        if (i < 1.0/3.0) return 2;
        else return 1;
    }
    if (((line1->att == TEXT)&&(line2->att == TEXT))){
        if ((i >= 0.5) || (a >= 0.5)) return 1;
        else return 0;
    }
    /*
    if (i >= a) {
        if (i < 1.0/3.0)&&(line1->att != TEXT) return 0;
        else return 1;
    } else {
        if (a < 1.0/3.0)&&(line return 0;
        else return 1;
    }
    */
}

/*.....Combine located line closest inside. the samaine.....*/
/*.....                                                    .....*/

void inside_set(first, head, line1=1)
struct linkblock *first;
struct linkblock *head;
struct linkblock **line1=1;
{
    int    n;
    struct samaline *tmp, *tmp1;

printf("before inside test\n");
for ( tmp = first; tmp != NULL; tmp = tmp->next){
    if (tmp->current == NULL) printf("null\n");
    else printf("index = %d x = %d y = %d l = %d u = %d \n",
                tmp->current->index, tmp->current->left_x,
                tmp->current->upper_y, tmp->current->length,
                tmp->current->width);
}

for (tmp = first; tmp != NULL; tmp = tmp->next){
    if (tmp->current == NULL) continue;
    for (tmp1 = tmp->next; tmp1 != NULL; tmp1 = tmp->next){
        if (tmp1->current == NULL) continue;
        if (tmp1->current->left_x > tmp->current->right_x) break;
        printf("index 1 = %d index 2 = %d \n", tmp->current->index, tmp1->current->index);
        if (a = index_of_combine(tmp->current, tmp1->current)) == 1){
            if ((tmp->current->att != TEXT)&&(tmp1->current->att != TEXT))){
```

【図104】

```
line_transfer(tmp->current, head, tmp1->current,
                                tmp->current->att);

    tmp->current = NULL;
    linelist[tmp->index] = NULL;
    if (!((tmp1->current->att&TEXT)
        tmp1->current->att = TITLE|TEXT;
    } else if (tmp1->current->att == TEXT){
        line_transfer(tmp1->current, head, tmp->current,
                                tmp1->current->att);

        tmp1->current = NULL;
        linelist[tmp1->index] = NULL;
        if (!(tmp->current->att&TEXT))
            tmp->current->att = TITLE|TEXT;
        break;
    }
} else if (n == 2) {
    if (tmp->current->att != TEXT) {
        line_transfer(tmp1->current, head, tmp->current,
                                tmp1->current->att);

        tmp1->current = NULL;
        linelist[tmp1->index] = NULL;
    } else {
        line_transfer(tmp->current, head, tmp1->current,
                                tmp->current->att);

        tmp->current = NULL;
        linelist[tmp->index] = NULL;
        break;
    }
}
*/
    }
}
printf("\n\n after inside test\n");
for ( tmp = first; tmp != NULL; tmp = tmp->next){
    if (tmp->current == NULL) printf("null\n");
    else printf("index = %d x = %d y = %d l = %d v = %d \n",
            tmp->current->index, tmp->current->left_x,
            tmp->current->upper_y, tmp->current->length,
            tmp->current->width);
}
}

/*****************************************************************/
/*                                                               */
/*      formation of title                                       */
/*                                                               */
/*****************************************************************/

void title_sentline_y_sort(  firstline, num)
struct linkblock **line_y_sort;
struct linkblock *firstline;
int     num;
{
    int     i, j, k;
    int     leftlevel, rightlevel;
    int     tmp, bottom;
    float   leftlength, rightlength;
    struct linkblock *tmp;
    struct sameline *first, *currentline, *topline;

printf("get into title_sent\n");
    for ( i = 0; i < num; i++){
        if (!(line_y_sort[i] == NULL)) continue;
        if (!(line_y_sort[i]->att != UNKNOWN)) continue;
        first = new_sameline(line_y_sort[i]);
        first->index = i;

currentline = first;
tmp = line_y_sort[i]->upper_y;
bottom = line_y_sort[i]->lower_y;
for ( j = i+1; j < num; j++){
    if (!(line_y_sort[j] == NULL)) continue;
    if (!(line_y_sort[j]->att != UNKNOWN)) continue;
    if (!(line_overlap(line_y_sort[i]->upper_y, line_y_sort[i]->lower_y,
                        tmp, bottom))){
        if (!((line_y_sort[i]->att == TEXT)||((line_y_sort[i]->att&TITLE)
            &&(line_y_sort[i]->length < MAXTOKU*SCALE_RATIO)))){
            currentline->next = new_sameline(line_y_sort[i]);
            currentline->next->previous = currentline;
            currentline = currentline->next;
            currentline->index = j;
            if (bottom < line_y_sort[i]->lower_y)
                bottom = line_y_sort[i]->lower_y;
        }
    } else if (line_y_sort[i]->upper_y > bottom) break;
}

for (j = i-1; j >= 0; j--){
    if (!(line_y_sort[j] == NULL)) continue;
    if (!(line_overlap(line_y_sort[j]->upper_y, line_y_sort[j]->lower_y,
                        tmp, bottom))){
        if (!((line_y_sort[j]->att == UNKNOWN)||
            (!((line_y_sort[j]->att==TEXT)||((line_y_sort[j]->att&TITLE)
                &&(line_y_sort[j]->length < MAXTOKU*SCALE_RATIO))))){
            currentline->next = new_sameline(line_y_sort[j]);
            currentline->next->previous = currentline;
            currentline = currentline->next;
            if (top > line_y_sort[j]->upper_y) top = line_y_sort[j]->upper_y;
            if (bottom < line_y_sort[j]->lower_y)
                bottom = line_y_sort[j]->lower_y;
        }
    } else
        if (line_y_sort[j]->lower_y < top-test)
            break;
}
sort_sameline(first);
inside_test(first, firstline, line_y_sort);

for (currentline = first; currentline != NULL;
                        currentline = currentline->next){
    if (currentline->current->NULL) continue;
    if (!((currentline->current->att&TEXT))||
        (currentline->current->att==UNKNOWN))){
        test_neighbor(currentline, &leftlevel, &rightlevel, &leftlength,
                        &rightlength, firstline, line_y_sort, NULL);

printf("index = %d leftlevel = %d rightlength= %f\n",
    currentline->current->index, leftlevel, rightlevel, leftlength, rightlength);

        if ((leftlevel >= 1)&&(rightlevel>=2)){
            currentline->current->att = TEXT|PICTURE|NEWTEXT;
        }
        if (leftlevel == 1){
        if ((line|((line|currentline->current->length/leftlength) < 1)(
            currentline->current->att = PICTURE|NEWTEXT|TEXT;
        } else {
            left_title(currentline, line_y_sort, firstline);
            test_right_neighbor(currentline, &rightlevel,
                                &rightlength, firstline, line_y_sort, NULL);
        }
        if (rightlevel == 1){
```

【図105】

```
    if ((int)((float)currentline->current->length/rightlength) > 1)M
        currentline->current->att = PICTURE|RIGHTJUSTIFIED;
    } else sigma_title(currentline, line_y_sort, direction);
}
if (((leftlevel >= 3)&&(rightlevel == 0))||
    ((leftlevel == 0)&&(rightlevel >= 3))||
    currentline->current->att & TEXT|PICTURE|ABSTRACT;
{
if ((leftlevel == 0)&&(rightlevel == 0)) {
    currentline->current->att = PICTURE|TEXT|ABSTRACT;
}
if (currentline->current->att == UNKNOWN)
    currentline->current->att = PICTURE|TEXT|PROCESS;

    currentline->current = NULL;
}
}
}
pack_line(line_y_sort, num);
sort_lnpointer(line_y_sort, *num);
}

/*************************************************************/
/*******  formation of text boundary                  *******/
/*************************************************************/

struct linkblock **boundary_form(line_x, num_bound, num)
struct linkblock **line_x;
*num_bound, *num;
int
{
    int     i, j;
    struct linkblock *tmpline;
    struct linkblock **bound;

    for (i=0, (*num_bound) = 0; i < *num; i++)
        if (line_x[i]->attachment) (*num_bound)++;
/* printf("numbound = %d\n", *num_bound); */
    if ((*num_bound) == 0) return NULL;

    bound = new_linkpointer(0, (*num_bound)-1);
    for (i = 0, j = 0; i < *num; i++)
    {
        if (line_x[i]->attachment){
            bound[j++] = line_x[i];
            line_x[i] = NULL;
        }
    }
    sort_lnpointer(bound, *num_bound);

    pack_line(line_x, num);
    return bound;
}

/*************************************************************/
/*******  combine two line                            *******/
/*************************************************************/

void linecombine(line1, line2)
struct linkblock *line1, *line2;
{

    if (line1->left_x <= line2->left_x)
        blisk_combine(line1, line2);
    else
        blisk_combine(line2, line1);
}

/*************************************************************/
/*******  separate formation of section block         *******/
/*************************************************************/

struct sectionblock *form_section(headsection, first, last, list_y_sort, column, thre
{
struct sectionblock *headsection;
int   first, last;
struct linkblock **list_y_sort;
int   column, thre;
{

    int   i,j, k, p;
    int   columnset, numsection, level;
    int   *hist_column;
    int   gapbegin[GAP], gapend[GAP];
    int   th;
    float alignsgn;
    float gap[GAP];
    struct sectionblock *psection, *section, *currentsection;
    struct linkblock   *tmp, *tmpline, *tmpline2;

    printf("get into form section\n");
    hist_column = (int*)calloc(9, column-1);
    for (i = 0; i < column; hist_column[i++] = 0);

    for (i = first; i <= last; i++){
        if (list_y_sort[i] == NULL) continue;
        for (j = list_y_sort[i]->left_x; j <= list_y_sort[i]->right_x; j++)
            hist_column[j]++;
    }

    for ( i = 0; i < GAP; i++){
        gapbegin[i]=gapend[i] = 0;
        gap[i] = 0.0;
    }

    /* column estimation */

/*  th = ((float)(list_y_sort[last]->lower_y-list_y_sort[first]->upper_y)/
                                        (float)(last-first+1.0);
*/
    th = (float)thre;
/*  th = 0.0;
*/
/*
printf(" first = %d last = %d th = %f\n", list_y_sort[first]->index,
                list_y_sort[last]->index, th);
for (i = 0; i < column; i++)
if (hist_column[i] != 0){
    printf("start from i = %d \n", i);
    for (j = i; j < column; j++)
        printf("%d ", hist_column[j]);
    break;
}
*/
    columnset = -1;
```

【図106】

```
for (i = 0; i < column; ){
    if ((float)hist_column[i] <= th){
        gapbegin[++columnset] = i;
        gap[columnset] = 0.0;
        for ( ; i < column&&(float)hist_column[i] <= th;
            gap[columnset] += (float)hist_column[i], i++);
        gap[columnset] /= (float)(i-gapbegin[columnset]);
        for ( j = gapbegin[columnset]; level < gap[columnset]; j--)
            if ((float)hist_column[j] > gap[columnset]){
                if (hist_column[j] <= level) {
                    level = hist_column[j];
                    continue;
                } else break;
            } else break;
        gapbegin[columnset] = j;
        for ( j = gapbegin[columnset]; j < gapend[columnset]; j++)
            if ((float)hist_column[j] > gap[columnset]){
                if (hist_column[j] <= level) {
                    level = hist_column[j];
                    continue;
                } else break;
            } else break;
        gapend[columnset] = j;
        if ((gapend[columnset]-gapbegin[columnset]) <= 2)
            columnset--;
        else if (columnset >= 2){
            if ((gapbegin[columnset]-gapend[columnset-1]-1)<(interval*2)){
                if ((gapend[columnset]-gapbegin[columnset])>
                    (gapend[columnset-1]-gapbegin[columnset-1])){
                    gapbegin[columnset-1] = gapbegin[columnset];
                    gapend[columnset-1] = gapend[columnset];
                }
                columnset--;
            }
        }
    if (columnset == CAP-1) break;
    else continue;
  }
  else
    i--;
}
/* group section */
nsection = new_eachblock(0, columnset);
for ( i = first; i <= last; i++){
    if (list_y_sort[i] == NULL) continue;
    for (j = 0, k = columnset; k>=
        if ((list_y_sort[i]->left_x > gapend[k]) &&
            (list_y_sort[i]->right_x< gapbegin[k+1])){
            put_section(list_y_sort[i], &(nsection[k]));
            j = 1;
            break;
        }
        if (j == 0)
            put_section(list_y_sort[i], &(nsection[columnset]));
    }
}
if (columnset > 0){
    mgend[current_reg].column_no = columnset;
    segment[current_reg].begin_colum = ivector(0, columnset);
    segment[current_reg].end_colum = ivector(0, columnset);
    for ( i = 0; i <= columnset; i++){
        segment[current_reg].begin_colum[i] = gapbegin[i];
        segment[current_reg].end_colum[i] = gapend[i-1];
    }
    for ( j = 1; j <= segment[current_reg].column_no; j++)
        printf("%d end = %d ", segment[current_reg].begin_colum[j],
                               segment[current_reg].end_colum[j]);
    printf("\n");
}
for (tapline = nsection[columnset].firstlineblock; tapline != NULL;
                                    tapline = tapline->next){
    for ( k = 0; k < columnset; k++)
        if ((tapline->left_x < gapbegin[k+1])&&
            (tapline->left_x > gapend[k]))
            break;
    if (k != columnset) {
        for (tap = nsection[k].firstlineblock; tap != NULL; ){
            if ((cover_range(tap->upper_y, tap->lower_y, tapline->upper_y,
                tapline->lower_y) > OVERLAP) ||
                (cover_range(tapline->upper_y, tapline->lower_y, tap->upper_y,
                tap->lower_y) > OVERLAP)){
                tapline = tap->next;
                take_section(tap, &(nsection[k]));
                if (tapright_x < tapline->left_x - text)
                    line_combine(tapline, tap)
                else
                    put_section(tap, &(nsection[columnset]));
                tap = tapline;
                continue;
            }
            tap = tap->next;
        }
    }
    for ( k = 0; k < columnset; k++)
        if ((tapline->right_x < gapbegin[k+1])&&
            (tapline->right_x > gapend[k]))
            break;
    if (k != columnset) {
        for (tap = nsection[k].firstlineblock; tap != NULL; ){
            if ((cover_range(tap->upper_y, tap->lower_y, tapline->upper_y,
                tapline->lower_y) > OVERLAP) ||
                (cover_range(tapline->upper_y, tapline->lower_y, tap->upper_y,
                tap->lower_y) > OVERLAP)){
                tapline = tap->next;
                take_section(tap, &(nsection[k]));
                if (tap->left_x < tapline->right_x + text)
                    line_combine(tapline, tap)
                else
                    put_section(tap, &(nsection[columnset]));
                tap = tapline;
                continue;
            }
            tap = tap->next;
        }
    }
}
/*
for ( i = 0; i <= columnset; i++){
    printf("section = %d \n", i);
```

【図107】

```
  for (tmp = section[i].firstlinkblock; tmp != NULL; tmp = tmp->next)
    printf("%d upper_y = %d lower_y = %d ", tmp->index, tmp->upper_y, tmp->lower_y);
  printf("\n");
}
*/

printf("text) = %f \n", text);
  alignmap = text*3.0;
printf("alignmap = %f \n", alignmap);
  numsection = 0;
  section = NULL;

  free((int *)hist_column);
  for (i = 0 ; i < columnmax; i++)
    {
      if (section[i].firstlinkblock==NULL) continue;
      currentsection = new_section(numsection);
      if (section == NULL) section = currentsection;
      currentsection->att = TEXT;
      for (tmp = section[i].firstlinkblock; tmp != NULL; ) {
        if (currentsection->numlinkblocks==0){
          currentsection->firstlinkblock = tmp;
          currentsection->currentlinkblock = tmp;
          tmp->previous = NULL;
          currentsection->numlinkblocks++;
          currentsection->upper_y = tmp->upper_y;
          currentsection->lower_y = tmp->lower_y;
          currentsection->left_x = tmp->left_x;
          currentsection->right_x = tmp->right_x;
          currentsection->att |= tmp->att;
          tmp = tmp->next;
          if (tmp != NULL)  tmp->previous->next = NULL;
        } else{
          if (((tmp->upper_y-currentsection->lower_y)>min(tmp->length,
              currentsection->currentlinkblock->length)) /* ){*/
                                                                    ||
/*
printf("create new section, tmp = %d att = %x currentsection->currentline = %d\n",tmp->index,tmp->att, currentsection->currentlinkblock->index);
*/
              currentsection = new_section(numsection);
              currentsection->att = TEXT;
            } else{
              if (currentsection->lower_y < tmp->lower_y)
                currentsection->lower_y = tmp->lower_y;
              if (currentsection->right_x < tmp->right_x)
                currentsection->right_x = tmp->right_x;
              if (currentsection->left_x > tmp->left_x)
                currentsection->left_x = tmp->left_x;
              if (lower_range(tmp->upper_y, tmp->lower_y,
                  currentsection->currentlinkblock->upper_y,
                  currentsection->currentlinkblock->lower_y,
                                                          0.5)){
                linkcombine(currentsection->currentlinkblock,
                            tmp);
                tmp = tmp->next;
              } else {
              currentsection->currentlinkblock->next = tmp;
              tmp->previous = currentsection->currentlinkblock;
              currentsection->currentlinkblock = tmp;
              currentsection->numlinkblocks++;
                                                    }
                                                  }
                                                }
                                              }
                                            }
                                          }
                                        }
                                      }
                                    }
                                  }
                                }
                              }
                            }
                          }
                        }
                      }
                    }
                  }
                }
              }
            }
          }
        }
      }
    }
  }
}

for (tmp = section[columnmax].firstlinkblock; tmp != NULL; ) {
  currentsection = new_section(numsection);
  currentsection->att = (ATT)0;
  if (section == NULL) section = currentsection;
  currentsection->currentlinkblock = tmp;
  currentsection->upper_y = tmp->upper_y;
  currentsection->lower_y = tmp->lower_y;
  currentsection->left_x = tmp->left_x;
  currentsection->right_x = tmp->right_x;
  tmp = tmp->next;
  if (tmp != NULL)  tmp->previous->next = NULL;
}

for (currentsection = section; currentsection != NULL;
                               currentsection = currentsection->next)
  printf(" x1 = %d y1 = %d x2 = %d y2 = %d att =%x \n", currentsection->left_x,
      currentsection->upper_y, currentsection->right_x,
      currentsection->lower_y, currentsection->att);

  free((struct partialblock *)section);
printf("end of freesection \n");
  return section;
}

void set_hasfield(first, last, line_y_sort, boundaryline)
  int first, last;
  struct linkblock **line_y_sort;
  struct linkblock *boundaryline;
{
  int i;
  int distance;

  distance = boundaryline->length;
  for ( i = first; i< last; i++){
    if((line_y_sort[i]->lower_y < boundaryline->lower_y)&&
                    (line_y_sort[i]->upper_y > boundaryline->upper_y-distance*1.5)){
      if (((line_y_sort[i]->left_x > boundaryline->left_x - text) &&
           (line_y_sort[i]->right_x < boundaryline->right_x + text))
          distance = boundaryline->upper_y - line_y_sort[i]->lower_y;
      if (distance < 0)  break;
    }
  }
  if ( distance <= 0) boundaryline->att |= TEXT;
  else {
    if (boundaryline->length >= text)
      boundaryline->att |= (TEXT|HOSEI);
    else
      boundaryline->att |= PICTURE; /*??? needs more consideration */
  }
}
```

【図108】

```
/*.....*/
/*.....*/
/*.....*/

void rm_section(head, section)
struct sectionblock *head, *section;
{

    if (head->currentmaintextsection == section){
        if (section->previous != NULL)
            section->previous->next = NULL;
        else
            head->firstmaintextsection = NULL;
    } else if (head->firstmaintextsection == section){
        head->firstmaintextsection = section->next;
        section->next->previous = NULL;
    } else {
        section->next->previous = section->previous;
        section->previous->next = section->next;
    }
}

/*.....*/
/*.....*/
/*.....*/

void rm_subsection(head, section)
struct sectionblock *head, *section;
{

    if (head->currentsubsection == section){
        if (section->previous != NULL)
            section->previous->next = NULL;
        else
            head->firstsubsection = NULL;
        head->currentsubsection = section->previous;
    } else if (head->firstsubsection == section){
        head->firstsubsection = section->next;
        section->next->previous = NULL;
    } else {
        section->next->previous = section->previous;
        section->previous->next = section->next;
    }
}

/*.....*/
/*.....*/
/*.....*/

void combinesec(first, second, head)
struct sectionblock *first, *second, *head;
{
    printf("combine %d => %d \n", second->index, first->index);
    first->firstinlineblock->previous = second->currentinlineblock;
    second->currentinlineblock->next = first->firstinlineblock;
    first->firstinlineblock = second->firstinlineblock;
    if (second->upper_y < first->upper_y)
        first->upper_y = second->upper_y;
    if (second->lower_y > first->lower_y)
        first->lower_y = second->lower_y;
    if (second->left_x < first->left_x)
        first->left_x = second->left_x;
    if (second->right_x > first->right_x)
        first->right_x = second->right_x;
    first->width = first->right_x - first->left_x + 1;
    first->length = first->lower_y - first->upper_y +1;
    first->numlines = first->numlines + second->numlines;
    first->numlineblock += second->numlineblock;

    if ((second->first_caption != NULL)&&(second->attr == (TEXT|CAPTION)))
        rm_section(head, second);
    else
        rm_insertion(head, second)
    free((struct sectionblock *)second);
}

/*.....*/
/*.....*/
/*.....*/

int boundarylin(first, topsec, bottomsec, align)
struct sectionblock *first, *topsec, *bottomsec;
int align;
{

    struct sectionblock *tmp;
    float topcover, bottcover, botcover, botyover;
    float yoverlap;
    int   x1, x2, y1, y2, x, y, count;
    unsigned char p;

    printf("get into boundary into\n");
    yoverlap = cover_range(bottomsec->upper_y, bottomsec->lower_y,
                           topsec->upper_y, topsec->lower_y);
    yoverlap = 0.0;
    if (topsec->attr&CAPTION){
        if (topsec->attr&CAPTION){
    printf("under_caption = %d x1 = %d x2 = %d \n",
        topsec->firstinlineblock->under_caption->index,
        topsec->firstinlineblock->under_caption->left_x,
        topsec->firstinlineblock->under_caption->right_x);
        if ((bottomsec->left_x <
            topsec->firstinlineblock->under_caption->left_x-text))||
            (bottomsec->right_x >
            topsec->firstinlineblock->under_caption->right_x+text))){
            return 1;
        }
```

【図109】

【図１１０】

```
    (tmp->right_x >= min(topsec->left_x, bottomsec->left_x)))
    return 0;

if (!(r_overlap(top->upper_y, top->lower_y,
        min(topsec->upper_y, bottomsec->upper_y),
            max(topsec->lower_y, bottomsec->lower_y)))
    if (!(r_overlap(top->left_x, top->right_x,
        min(topsec->left_x, bottomsec->left_x),
            max(topsec->right_x, bottomsec->right_x)))
        return 1;
    }
}
*/
    return 0;
}

/*.....*/
/*..... Test if some section fully inside the other section .....*/
/*.....*/

int full_inside(first, second)
struct sectionblock *first, *second;
{
    if (((first->left_x >= second->left_x)&&(first->right_x <= second->right_x)&&
        ((first->upper_y >= second->upper_y)&&(first->lower_y <= second->lower_y)))
        return 1;
    else
        return 0;
}

/*.....*/
/*..... Test if two section are neighbors (belong to same column) .....*/
/*.....*/

int same_section(sort_y, first, sec, num)
struct sectionblock **sort_y;
first, sec, num;
{
    int   x1, y1, x2, y2, i, j, count;
    int   in_segment, in_column, in_segment, in_column;
    struct sectionblock *tmp1, *tmp2;
    int   area1_x1, area1_y1, area1_y2;
    int   area2_x1, area2_x2, area2_y2;
    char  p;
printf("in same_section(sort_y, %d, %d)\n", sort_y[first]->index,
        sort_y[sec]->index);

    if (sort_y[first]->left_x < sort_y[sec]->left_x){
        tmp1 = sort_y[first];
        tmp2 = sort_y[sec];
    } else {
        tmp1 = sort_y[sec];
        tmp2 = sort_y[first];
    }

    if (!(r_overlap(tmp1->upper_y, tmp1->lower_y,
            tmp2->upper_y, tmp2->lower_y) == 0)) return 0;

    if ((tmp1->attrCAPTION)&&(tmp2->attrCAPTION)){
        if (tmp1->firstinblock->node_capsize !=
            tmp2->firstinblock->node_capsize){
            return 0;
        }
    } else if ((tmp1->attrCAPTION)||(tmp2->attrCAPTION)
        return 0;

    in_segment = tmp1->boundary_index;
    in_column = 0;
    in_segment = tmp2->boundary_index;
    in_column = 0;

/*  for (j = 1, i = in_segment; j <= segment[i].column_no; j++){
        printf("%d, %d, %d, %d, column_no1 = %d column_no2 = %d \n", j,
        tmp1->left_x, tmp1->right_x, segment[i].begin_column[j], segment[i].end_column[j]);
*/
        if (cover_range(tmp1->left_x, tmp1->right_x,
            segment[i].begin_column[j], segment[i].end_column[j]))==-1.0){
            in_column = j;
            break;
        }
    }

/*  printf("segment = %d coll = %d seg2 = %d col2 = %d\n", in_segment, in_column,
        in_segment, in_column);
*/
    for (j = 1, i = in_segment; j <= segment[i].column_no; j++){
        if (cover_range(tmp2->left_x, tmp2->right_x,
            segment[i].begin_column[j], segment[i].end_column[j]))==-1.0){
            in_column = j;
            break;
        }
    }

/*  printf("segment = %d coll = %d seg2 = %d col2 = %d\n", in_segment, in_column,
        in_segment, in_column);
*/
    if ((in_segment >= 0)&&(in_segment >= 0)){
        if ((in_segment != in_segment))
            {
printf("different segments\n");
            if (((in_column>0)&&(in_column>0))){
                if ((cover_range(tmp1->left_x, tmp2->right_x,
                    segment[in_segment].begin_column[in_column],
                    segment[in_segment].end_column[in_column]))!=0 )&&
                    (cover_range(tmp1->left_x, tmp2->right_x,
                    segment[in_segment].begin_column[in_column],
                    segment[in_segment].end_column[in_column]))!=1.0)){
                    return 0;
                }
            } else
                return 0;
        } else {
            if ((tmp2->left_x<tmp2->right_x)&&(tmp1->left_x<tmp1->right_x)){
                if ((tmp1->width > (int)(tmp1*3.0))||
                    (tmp2->width > (int)(tmp1*3.0))){
                    return 0;
                }
                printf("same segment\n");
            } else
                return 0;
        }
/* printf("same segment, same column\n");*/
        x1 = tmp1->right_x;
        x2 = tmp2->left_x;
        if (tmp1->upper_y != tmp2->upper_y){
```

【図111】

```
    y1 = tmp1->upper_y;
    area1_y1 = tmp1->left_x;
    area1_y2 = tmp1->right_x;
    area2_y1 = y1;
    area2_y2 = tmp1->upper_y-1;
} else {
    y1 = tmp1->upper_y;
    area1_y1 = tmp1->right_x+1;
    area1_y2 = tmp2->right_x;
    area2_y1 = y1;
    area2_y2 = tmp2->upper_y-1;
}

if (tmp1->lower_y == tmp2->lower_y){
    y2 = tmp1->lower_y;
    area1_x1 = tmp1->left_x;
    area1_x2 = tmp1->right_x-1;
    area2_x1 = tmp2->left_x;
    area2_x2 = tmp2->right_x;
    area1_y1 = tmp1->lower_y+1;
    area1_y2 = tmp1->lower_y;
} else {
    y2 = tmp1->lower_y;
    area1_x1 = tmp1->left_x;
    area1_x2 = tmp1->right_x-1;
    area1_y1 = tmp1->lower_y+1;
    area1_y2 = tmp1->lower_y;
}

p = ****(((0=0));

if ((in_segment == in_segment)&&(in_column != in_column))
    return 0;
if (((in_segment != in_segment) ||((in_column != in_column))&&
    if (((x2-x1)>(int)sect/2)&&((tmp2->width > (int)(2.0*sect)))&&
            (tmp2->width > (int)(2.0*sect)))) return 0;

for ( i = x1; i <= x2; i++)
    for (j = y1; j <= y2; j++)
        if ((original_p[i][j]&(1<<p)) return 0;

/* printf("no test between"); */
  if ((x2-x1) < d)||((in_segment == in_segment)&&(in_column == in_column)
        d)||((tmp1->left_x-tmp1->right_x)<(int)(text*1.2))){
/* printf("narrow gap \n"); */
    for (i = area1_x1; i <= area1_x2; i++){
        for (j = area1_y1; j <= area1_y2; j++){
            if ((original_p[i][j]&(1<<p)) return 0;
        }
    }
/* printf("place \n"); */
    for (i = area2_x1; i <= area2_x2; i++){
        for (j = area2_y1; j <= area2_y2; j++){
            if ((original_p[i][j]&(1<<p)) return 0;
        }
    }
    return 1;
} else return 0;
}

/*****************************************************************/
/*********  test if at test block overlap with the two combined block  *********/
/*****************************************************************/

int test_intersect(headsec, first, second)
struct sectionblock *headsec;
struct sectionblock *first, *second;
{
    struct sectionblock *sect;
    int x1, y1, x2, y2;
    float xcover, ycover, xcover2;

/*  printf("get into test inter first = %d second = %d  \n",
                (first->index, second->index));
*/
    if ((first->upper_y > second->upper_y){
        tmpsec = first;
        first = second;
        second = tmpsec;
    }
    x1 = min(first->left_x, second->left_x);
    x2 = max(first->right_x, second->right_x);
    y1 = min(first->upper_y, second->upper_y);
    y2 = max(first->lower_y, second->lower_y);

    for (tmpsec = headsec->first;tmpsec != NULL; tmpsec = tmpsec->next){

/*
printf("%d x1 = %d y1 = %d ", tmpsec->index, tmpsec->left_x, tmpsec->upper_y);
*/
        if ((tmpsec == first)||(tmpsec == second)) continue;
        if ((tmpsec->upper_y > y2)||(tmpsec->lower_y < y1)) continue;
        xcover = cover_range(tmpsec->left_x, tmpsec->right_x, x1, x2);
        if (xcover == 0.0) continue;
        xcover2 = cover_range(tmpsec->left_x, tmpsec->right_x, x1-(int)text1,
                              x2-(int)text1);
/*
printf("%d x1 = %d y1 = %d ", tmpsec->index, tmpsec->left_x, tmpsec->upper_y);
*/
        if ((tmpsec->upper_y < first->lower_y)&&
            (tmpsec->lower_y > second->upper_y)) return 1;
        if ((tmpsec->upper_y < first->lower_y)||
            (tmpsec->lower_y > second->upper_y)){
            if ((xcover2>0.0)||(xcover>0.0)){
                y2 = tmpsec->lower_y;
                if (tmpsec->left_x < x1) {
                    x1 = tmpsec->left_x;
                }
                if (tmpsec->right_x > x2) {
                    x2 = tmpsec->right_x;
                }
                continue;
            }
            else return 1;
        }
/* printf("xcover2 = %f xcover = %f \n",xcover2, xcover); */
        if ((tmpsec->upper_y > first->upper_y-(int)text1)||
            (xcover2>0.0)||(xcover>0.0)) {
            if (tmpsec->upper_y < y1)
                y1 = tmpsec->upper_y;
            if (tmpsec->left_x < x1) {
                x1 = tmpsec->left_x;
            }
            if (tmpsec->right_x > x2) {
                x2 = tmpsec->right_x;
            }
            continue;
        }
        else return 1;
    }
    if (x1 < min(first->left_x, second->left_x))
    second->left_x = x1;
    if (x2 > max(first->right_x, second->right_x))
    second->right_x = x2;
```

【図１１２】

```
    if (y1 < min(first->upper_y, second->upper_y))
        first->upper_y = y1;
    if (y2 > max(first->lower_y, second->lower_y))
        second->lower_y = y2;
    return 0;
}

/*****************************************************************/
/******** Test if the text stick with the horizontal line ********/
/*****************************************************************/

void section_combine(first, boundary)
struct sectionblock *first;
struct lineblock    *boundary;
{
    int   i, j, k, sectionnum, align, if_combine, num;
    float alignmp;
    struct sectionblock **sort_y_section;

printf("get into section combine \n");

    sectionnum = (int)((float)text1*1.3);
    alignmp = (text1*2.0;
    sort_y_section = sort_section(first, &num);

printf("after sort_y section, num = %d \n", num);

for (i = 0; i < num; i++)
printf("i = %d index = %d x1 = %d y1 = %d x2 = %d y2 = %d att = %x bound = %d\n", /*
        i, sort_y_section[i]->index,
    sort_y_section[i]->left_x, sort_y_section[i]->upper_y, sort_y_section[i]->righ
t_x,                                                                    */
    sort_y_section[i]->lower_y, sort_y_section[i]->att,
    sort_y_section[i]->boundary_indeed);

    for (i = 0; i < num; i++){
printf("i = %d \n", i, sort_y_section[i]->index);
    if (sort_y_section[i]->att == (TEXTTITLE)) continue;
    for (j = i+1; j < num; j++){
        if (sort_y_section[j] == NULL){
            sort_y_section[j]->att == (TEXTTITLE)) continue;
        if_combine = 0;
        if (!full_inside(sort_y_section[i], sort_y_section[j])){
            if_combine = 1;
        }
        else {
            if ((sort_y_section[i]->upper_y <= sort_y_section[j]->lower_y+
                    (int)text1*1.5)){

                align=sort_y_section[i]->fir_titleblock->length,
                sort_y_section[j]->current_titleblock->length)){
        if ((((_overlap(sort_y_section[i]->left_x, sort_y_section[i]->right_x,
                sort_y_section[j]->left_x, sort_y_section[j]->right_x))){
                align = 0;
                if (sort_y_section[j]->left_x < sort_y_section[i]->left_x)
                align |= LEFTBOTTOMCOR;
                else if (sort_y_section[i]->left_x > sort_y_section[j]->left_x)
                align |= LEFTTOPCOR;
            if ((((float)abs(sort_y_section[i]->left_x -
                        sort_y_section[j]->left_x) <= alignmp)
                align |= LEFTALIGN;
            if (sort_y_section[i]->right_x > sort_y_section[j]->right_x)
                align |= RIGHTBOTTOMCOR;
            else if (sort_y_section[i]->right_x < sort_y_section[j]->right_x)
                align |= RIGHTTOPCOR;
            if ((((float)abs(sort_y_section[i]->right_x -
                        sort_y_section[j]->right_x) <= alignmp)
                align |= RIGHTALIGN;
            if (sort_y_section[i]->left_x > sort_y_section[j]->left_x)
                align |= LEFTALIGN;
            if (((float)abs(sort_y_section[i]->left_x -
                        sort_y_section[j]->left_x) <= alignmp)
                align |= LEFTALIGN;
            if (sort_y_section[i]->right_x < sort_y_section[j]->right_x)
                align |= RIGHTTOPCOR;
            else if (sort_y_section[i]->right_x > sort_y_section[j]->right_x)
                align |= RIGHTBOTTOMCOR;
            if ((((float)abs(sort_y_section[i]->right_x -
                        sort_y_section[j]->right_x) <= alignmp)
                align |= RIGHTALIGN;
            if (((align&LEFTALIGN) == RIGHTALIGN)){
                switch(align&LEFTALIGN){
                case LEFTALIGN:
                    if_combine = 1;
                    break;
                case LEFTTOPCOR:
                    if_combine = 0;
                    break;
                case LEFTBOTTOMCOR:
                    if (i==sort_y_section[1] == 1)
                        if_combine = 1;
                    else
                        if_combine = 0;
                    break;
                }
```

【図１１３】

【図１１４】

```
/*.....*/
/*.....test if located within the hpicture.....*/
/*.....*/

int if_hpicture(tmp1, tmp2)
struct linkblock *tmp1;
struct sectionblock *tmp2;
{
    int x, y, count;
    unsigned char p1;

    p = TEXT[TATTOOU];
    for (y = tmp1->upper_y; y <= tmp1->lower_y; y++){
        count = 0;
        for (x = tmp1->left_x-1; x >= tmp2->left_x; x--)
            if ((original_p[y][x] & 0x80) count++;
/* printf("count 1 = %d \n", count); */
        if (count <= ((int)(HPOINT/SCALE_RATIO)-2)) continue;
        count = 0;
        for (x = tmp1->right_x+1; x <= tmp2->right_x; x++)
            if ((original_p[y][x] & 0x80) count++;
/* printf("count 2 = %d \n", count); */
        if (count > ((int)(HPOINT/SCALE_RATIO)-2)) return 1;
    }

    for (x = tmp1->left_x; x <= tmp1->right_x; x++){
        count = 0; tmp1->upper_y[x] = y = tmp2->upper_y; y--)
        for ( ; y > tmp1->lower_y[x]; y--)
            if ((original_p[y][x] & 0x80) count++;
        if (count <= ((int)(HPOINT/SCALE_RATIO)-3)) continue;
        count = 0;
        for ( ; y > tmp1->lower_y[x]; y--)
            if ((original_p[y][x] & 0x80) count++;
        if (count >= ((int)(HPOINT/SCALE_RATIO)-3)) return 1;
    }
    return 0;
}

/*.....*/
/*.....test if text line has some neighbor in some section.....*/
/*.....*/

int no_line_neighbor(tmp1, tmp2)
struct sectionblock *tmp1;
struct linkblock *tmp2;
{
    struct linkblock *tmp_line;

printf("tmp1 = %d x1 = %d y1 = %d x2 = %d y2 = %d \n", tmp1->index,
        tmp1->left_x, tmp1->upper_y, tmp1->right_x, tmp1->lower_y);
    for (tmp_line = tmp2->firstlinkblock; tmp_line != NULL;
                tmp_line = tmp_line->next){
        if (tmp_line == tmp1) continue;
printf("tmp_line = %d x1 = %d y1 = %d x2 = %d y2 = %d \n", tmp_line->index,
        tmp_line->left_x, tmp_line->upper_y, tmp_line->right_x,tmp_line->lower_y);
        if (((if_overlap(tmp_line->upper_y-(int)(text*1.5),
            tmp_line->lower_y+(int)(text*1.5), tmp1->upper_y, tmp1->lower_y)!=0
            (if_overlap(tmp_line->left_x-tmp_line->length,
                tmp_line->right_x+tmp_line->length,
                tmp1->left_x, tmp1->right_x))!
            if ((tmp_line->length >= (int)(text*2.5))&&
                (tmp_line->width >= (int)(text*2.5))) return 0;
        }
    }
    return 1;
}
```

【図115】

```
        break;
    } else {
        if ((tmpn1->attribute&PICTURE)&&(xcover == 1.0)&&
            (tmpn1->length*0.5<length)){
            rearrange = 0;
            for(tmp1 = tmp->firstinablock; tmp1!= NULL; ){
                within_hple = 0;
                if ((tmp1->width < (int)(ecc1*4))||
                    (tmp1->length < (int)(ecc1/2))||&(no_line_neighbor(tmp1, tmp1
/*
!!!(
*/
            if ((tmpn1->attribute&PICTURE)&&(xcover == 1.0)&&(ycover == 1.0)){
                rearrange = 0;
                for(tmp1 = tmp->firstinablock; tmp1!= NULL; ){
                    within_hple = 0;
                    within_hple = if_inhpicture(tmp1, tmpn1);
                    if (((tmp1->width < (int)(ecc1*4))||
                        (tmp1->length < (int)(ecc1/2)))&&(no_line_neighbor(tmp1, tmp1
!!!(
*/
            if((within_hple == 1)&&(tmp1->numunblock <= 2))){ */
                    tmp13 = tmp1->next;
                    take_section(tmp1, tmp1);
                    bike_combine(tmpn1->firstinablock, tmp1);
                    tmp1 = tmp13;
                    rearrange = 1;
                } else tmp1 = tmp1->next;
            } else continue;
/* printf("rearrange = %d !!", rearrange); */
/* printf("tmp1->firstinablock = %08x", tmp1->firstinablock); */
            if (rearrange == 1){
                if (tmp1->firstinablock == NULL)
                    fn_section(firstsection, tmp1);
                else {
                    tmp1->upper_y = tmp1->firstinablock->upper_y;
                    tmp1->lower_y = tmp1->firstinablock->lower_y;
                    tmp1->left_x = tmp1->firstinablock->left_x;
                    tmp1->right_x = tmp1->firstinablock->right_x;
                    for (tmp = tmp1->firstinablock->next;
                         tmp1 != NULL; tmp1 = tmp1->next){
                        if (tmp1->left_x < tmp1->left_x)
                            tmp1->left_x = tmp1->left_x;
                        if (tmp1->width > tmp1->right_x)
                            tmp1->right_x = tmp1->left_x;
                        if (tmp1->upper_y < tmp1->upper_y)
                            tmp1->upper_y = tmp1->upper_y;
                        if (tmp1->lower_y > tmp1->lower_y)
                            tmp1->lower_y = tmp1->lower_y;
                    }
                    tmp1->length = tmp1->lower_y-tmp1->upper_y+1;
                    tmp1->width = tmp1->right_x-tmp1->left_x+1;
/* printf("after tmp%d\n"); */
                    tmpn1->left_x = tmpn1->firstinablock->left_x;
                    tmpn1->right_x = tmpn1->firstinablock->right_x;
                    tmpn1->upper_y = tmpn1->firstinablock->upper_y;
                    tmpn1->lower_y = tmpn1->firstinablock->lower_y;
                    tmpn1->length = tmpn1->lower_y-tmpn1->upper_y+1;
                    tmpn1->width = tmpn1->right_x-tmpn1->left_x+1;
                }
                no_sec = 1;
                tmp1 = tmp2;
                break;
            }
        }
    }
    if (no_sec == 0) tmp1 = tmp1->next;
}
for (tmpn1 = firstsection->firstcontentssection; tmpn1 != NULL;
                                                 tmpn1 = tmpn1->next){
    for ( tmpn2 = tmpn1->next; tmpn2 != NULL; tmpn2 = tmpn2->next){
        if (tmpn2->upper_y > tmpn1->lower_y) break;

        cover1_x = cover_range(tmpn1->left_x, tmpn1->right_x,
        cover2_x = cover_range(tmpn2->left_x, tmpn2->right_x,
        cover1_y = cover_range(tmpn1->upper_y, tmpn1->lower_y,
        cover2_y = cover_range(tmpn2->upper_y, tmpn2->lower_y,
                               tmpn1->upper_y, tmpn1->lower_y);
        xcover2 = cover_y_nge(tmpn2->left_x, tmpn2->right_x,
                              tmpn1->left_x,tmpn1->length/2,
                              tmpn1->right_x*tmpn1->length/2);

        if ((tmpn1->attribute&&(tmpn2->attribute&PICTURE)&&(xcover2>0.6)){
            tmp = tmpn1->previous;
            fcombineone(tmpn1, tmpn2, firstsection);
            tmpn12 = tmp;
            continue;
        } else if ((cover1_x == 0.0)||(cover1_y == 0.0)) continue;

        v = tmpn2->attribute;
        combine = 0;
        if (tmpn1->attribute){
            if (!v == PICTURE)||(v == (PICTURE|TEXT))||.
                ((v&HOLE)||(v&SOLID))){
                    combine = 1;
                } else if ((cover2_x == 1.0)&&(cover2_y == 1.0)){
                    combine = 1;
                } else continue;
        }
        if (tmpn1->att == PICTURE){
            if (v == (TEXT|PICTURE)){
                if (tmpn2->width < (int)(ecc1*3)
                    tmpn2->att = PICTURE;
            } else if ((v&SOLID)||(v&VOID))
                if ((cover2_x == 1.0)&&(cover2_y == 1.0))
                    tmpn2->att = PICTURE;
            continue;
        }
        if ((tmpn1->attribute(TEXT|PICTURE))==(TEXT|PICTURE)){
            if (tmpn1->att == PICTURE)
                tmpn1->att = PICTURE;
                combine = 2;
            } else {
                if (v == PICTURE){
                    if (tmpn2->width < (int)(ecc1*3))
                        tmpn1->att = PICTURE;
                }
                continue;
            }
        }
        if ((tmpn1->attribute&PICTURE)){
            if (!v&(TEXT|PICTURE)&&(cover2_y > 0.5)){
                tmp = tmpn2->previous;
                fcombineone(tmpn1, tmpn2, firstsection);
                tmpn12 = tmp;
                continue;
            }
        }
    }
```

【図116】

```
if ((next->attribute[0]||next->attribute[1]){
  if ((v == MAXCOVERAGE(cover_x == 1.0)&&(cover_y == 1.0)){
    combine = 2;
    next->att = MAXCOVER;
  } else{
    if ((v == MAXCOVERAGE(cover_x == 1.0)&&(cover_y == 1.0))
      next->att = v;
      continue;
    }
  }
  if (combine != 0){
/* printf("combine %d and %d \n", next->index, next1->index); */
    if (combine == 1)
      bline_combine(next->firstlineblock, next2->firstlineblock);
    else if (combine == 2)
      line_combine(next->firstlineblock, next2->firstlineblock);
    tmp = next2->prev;
    re_intersection(firstsection, next2);
    next2 = tmp;
    next->left_x = next->firstlineblock->left_x;
    next->right_x = next->firstlineblock->right_x;
    next->upper_y = next->firstlineblock->upper_y;
    next->lower_y = next->firstlineblock->lower_y;
    next->length = next->lower_y-next->upper_y+1;
    next->width = next->right_x-next->left_x+1;
  }
}

/*****************************************************************/
/*******  Section formation for each head line block  ************/
/*****************************************************************/

void section_block(firstsection, firstline, column)
struct sectionblock *firstsection;
struct linablock  *firstline;
int  column;
{
  int   i, j, numline;
  int   firstn, lastn, secondres;
  int   num_bound, numsection, numcontsection;
  struct linablock *line_y_sort, **boundary;
  struct linablock *tmpline;
  struct sectionblock *tsection, *subcurrentsec;

  printf("get into section block index = %d y=%d \n", firstline->index,
          firstline->left_x, firstline->upper_y);
    sby = firstline->upper_y;
    eby = firstline->lower_y;
    tsecl = (firstline->right_x-firstline->left_x+1)*(sby-eby+1);
          int_length(firstline->fin_block->avg_height);
/*    tsecl = sw_dis_length(firstline); */
  printf("tsecl = %f \n", tsecl);
  line_y_sort = sort_y(firstline, &numline);
  printf("after line_y_sort numline = %d \n", numline);
  title_semiline_y_sort, firstline, &numline);
  printf("after title_sem numline = %d \n", numline);
  boundary = boundary_form(line_y_sort, &num_bound, &numline);
  printf("after boundary_form numline = %d num_bound = %d \n", numline, num_bound);

  segment = new_segment(0, num_bound);
  current_seg = 0;
  firstn = lastn = numfirst = 0;
  for (numsection = 0; numsection<numsection = 0; i = 0;
                                                  i < num_bound; i++){

    printf("boundary = %d \n", boundary[i]->index;
                                  current_seg = i;
                                  firstn = lastn = next=first;
                                  for ( j = firstn ; j < numline;j&
                                    (line_y_sort[j]->lower_y <= boundary[i]->upper_y); lastn = j, j++);
                                    next=first = j;
                                                        /*
    if ((v == MAXCOVERAGE(cover_x == 1.0)&&(cover_y == 1.0)){
      next->att = v;
      continue;
    }
  }
  if (combine != 0){
/* printf("combine %d and %d \n", next->index, next1->index); */
    if (combine == 1)
      bline_combine(next->firstlineblock, next2->firstlineblock);
    else if (combine == 2)
      line_combine(next->firstlineblock, next2->firstlineblock);
    tmp = next2->prev;
    re_intersection(firstsection, next2);
    next2 = tmp;
    next->left_x = next->firstlineblock->left_x;
    next->right_x = next->firstlineblock->right_x;
    next->upper_y = next->firstlineblock->upper_y;
    next->lower_y = next->firstlineblock->lower_y;
    next->length = next->lower_y-next->upper_y+1;
    next->width = next->right_x-next->left_x+1;
  }
}

/*****************************************************************/
/*******  Section formation for each head line block  ************/
/*****************************************************************/

void section_block(firstsection, firstline, column)
struct sectionblock *firstsection;
struct linablock  *firstline;
int  column;
{
  int   i, j, numline;
  int   firstn, lastn, secondres;
  int   num_bound, numsection, numcontsection;
  struct linablock *line_y_sort, **boundary;
  struct linablock *tmpline;
  struct sectionblock *tsection, *subcurrentsec;

  printf("get into section block index = %d y=%d \n", firstline->index,
          firstline->left_x, firstline->upper_y);
    sby = firstline->upper_y;
    eby = firstline->lower_y;
    tsecl = (firstline->right_x-firstline->left_x+1)*(sby-eby+1);
          int_length(firstline->fin_block->avg_height);
/*    tsecl = sw_dis_length(firstline); */
  printf("tsecl = %f \n", tsecl);
  line_y_sort = sort_y(firstline, &numline);
  printf("after line_y_sort numline = %d \n", numline);
  title_semiline_y_sort, firstline, &numline);
  printf("after title_sem numline = %d \n", numline);
  boundary = boundary_form(line_y_sort, &num_bound, &numline);
  printf("after boundary_form numline = %d num_bound = %d \n", numline, num_bound);

  segment = new_segment(0, num_bound);
  current_seg = 0;
  firstn = lastn = numfirst = 0;
  for (numsection = 0; numsection<numsection = 0; i = 0;
                                                  i < num_bound; i++){

    printf("boundary = %d \n", boundary[i]->index;
                                  current_seg = i;
                                  firstn = lastn = next=first;
                                  for ( j = firstn ; j < numline;j&
                                    (line_y_sort[j]->lower_y <= boundary[i]->upper_y); lastn = j, j++);
                                    next=first = j;
                                                        /*
    if ((boundary[i]->att == (TEXT|TITLE)){
      subcurrentsec = new_section(firstsection);
      subcurrentsec->index = ++numsection;
    } else {
        subcurrentsec = new_section(firstsection);
        subcurrentsec->index = ++numsection;
      }
    }
    if (boundary[i]->att == (TEXT|TITLE)){
      subcurrentsec = new_section(firstsection);
      subcurrentsec->index = ++numcontsection;
    } else {
        subcurrentsec = new_section(firstsection);
        subcurrentsec->index = ++numsection;
      }
    }
    subcurrentsec->firstlineblock =
                  subcurrentsec->currentlineblock = boundary[i];
    subcurrentsec->upper_y = boundary[i]->upper_y;
    subcurrentsec->lower_y = boundary[i]->lower_y;
    subcurrentsec->left_x = boundary[i]->left_x;
    subcurrentsec->right_x = boundary[i]->right_x;
    subcurrentsec->width = boundary[i]->width;
    subcurrentsec->length = boundary[i]->length;
    subcurrentsec->att = boundary[i]->att;
    subcurrentsec->att &= ~(UNKNOWN);
    boundary[i]->previous = NULL;
    boundary[i]->next = NULL;
  }

  printf("next=first = %d numline = %d\n", next=first, numline);
    if (next=first < numline) {
      curren_seg = num_bound;
      firstn = next=first;
      lastn = numline-1;
      subcurrentsec = form_section(firstsection, firstn, lastn,
                                   line_y_sort, column, 1);
      for (tsection = subcurrentsec; tsection != NULL;
                                     tsection = tsection->next){
        tsection->width = tsection->right_x - tsection->left_x+1;
        tsection->length = tsection->lower_y-tsection->upper_y+1;
        tsection->index = ++numsection;
        tsection->boundary_index = current_seg;
      }
    }

  if (firstsection->firstsubsection != NULL){
    printf("before attribute_test \n");
                attribute_test(firstsection);
    printf("after attribute_test \n");
    for ( i = 0; i <= num_bound; i++){
      printf("segment = %d no_seg = %d ", i, segment[i].column_no);
```

【図117】

```
        for ( j = 1; j <= segment[i].colum_no; j++)
            printf("bg = %d end = %d ", segment[i].begin_column[j],
                                        segment[i].end_column[j]);
        printf("\n");
    }

/*      section_separate(firstsection); */

printf("before section combine\n");
/*
    section_combine(firstsection, boundary);
*/

    section_combine(firstsection, boundary);


    inside_text(firstsection);


    inside_text(firstsection);


  }
    free((struct segment_gap *)segment);
    free((struct lineblock **)line_y_sort);
    free((struct lineblock **)boundary);

/*
    section_combine(firstsection, boundary);
*/
/*    inner_section_combine(firstsection); */
}

/*************************************************************************/
/********* Table test                                           *********/
/*************************************************************************/
void  move_line_to_section(firstsection, firstline)
struct sectionblock *firstsection;
struct lineblock    *firstline;
{
    struct lineblock *tmpline, *tmpl1;
    struct sectionblock *currentsection;
    int i;

printf("get into move \n");
    for (i = 0, tmpline = firstline->firstsubline, tmpline != NULL; )
        {
            currentsection = new_section(firstsection);
            currentsection->firstlineblock = currentsection->currentlineblock
                                           = tmpline;
            currentsection->left_x = tmpline->left_x;
            currentsection->right_x = tmpline->right_x;
            currentsection->upper_y = tmpline->upper_y;
            currentsection->lower_y = tmpline->lower_y;
            currentsection->width = tmpline->width;
            currentsection->length = tmpline->length;
            currentsection->index = ++i;
            currentsection->att = TEXT;
            firstsection->numlineblock++;
            tmpl1 = tmpline->next;
            tmpline->next = NULL;
            tmpline->previous = NULL;
            tmpline = tmpl1;
        }

printf("before end\n");
    for (i = 0, tmpline = firstline->firstnontextline; tmpline != NULL; )
            {
                currentsection = new_ntsection(firstsection);
                currentsection->firstlineblock = currentsection->currentlineblock
                                               = tmpline;
                currentsection->left_x = tmpline->left_x;
                currentsection->right_x = tmpline->right_x;
                currentsection->upper_y = tmpline->upper_y;
                currentsection->lower_y = tmpline->lower_y;
                currentsection->width = tmpline->width;
                currentsection->length = tmpline->length;
                currentsection->index = ++i;
                currentsection->att = tmpline->att;
                tmpl1 = tmpline->next;
                tmpline->next = NULL;
                tmpline->previous = NULL;
                tmpline = tmpl1;
            }
/*
printf("in move \n");
for (currentsection=firstsection->firstsubsection, currentsection != NULL;
                                    currentsection = currentsection->next)
printf("index = %d x = %d y = %d \n", currentsection->index, currentsection->left_x,
                                    currentsection->upper_y);
for (currentsection=firstsection->firstnontextsection; currentsection != NULL;
                                    currentsection = currentsection->next)
printf("index = %d x = %d y = %d \n", currentsection->index, currentsection->left_x,
                                    currentsection->upper_y);
*/
printf("get out of move \n");
}

/*************************************************************************/
/********* Test if the text stick with the horizontal line      *********/
/*************************************************************************/

void ini_section_block(firstsection, firstline, p, column)
struct sectionblock *firstsection;
struct lineblock *firstline;
unsigned char **p;
int     column;
{
    struct sectionblock *tmpsection;
    int     num;
    struct lineblock *tmpline;


    original_p = p;
    firstsection->firstlineblock = firstsection->currentlineblock = firstline;
/*printf("firstline index = %d text = %d nontext = %d group = %d \n",
        firstline->index,   firstline->firstsubline, firstline->firstnontextline,
        firstline->group); */
    if ((firstline->firstsubline != NULL)||((firstline->firstnontextline != NULL))||
        if ((firstline->att&TABLE)||((firstline->att&PICTURE)&&
                        (firstline->group >= 2)&&(!(firstline->att&FRAME))))
            move_line_to_section(firstsection, firstline);
        else
            section_block(firstsection, firstline, column);

        firstline->firstsubline = NULL;
        firstline->firstnontextline = NULL;

    }


    for (tmpsection = firstsection->firstnontextsection; tmpsection != NULL;
```

【図119】

```
/*****************************************************************/
/*                                                               */
/*                        File: table.c                          */
/*                      Author: Shin-Yuan Wang                   */
/*                        Date : 3-6-92                          */
/*                                                               */
/*****************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
#include "table.h"

extern void exit();

/*****************************************************************/
/*                                                               */
/*  alloate new white_for_table structure                        */
/*                                                               */
/*****************************************************************/

struct white_for_table *new_wtable(whitab)
struct whitablock *whitab;
{
    struct white_for_table *v;

    v = (struct white_for_table *)malloc(sizeof(struct white_for_table));
    if (v == NULL){
        printf("error: in new white for table\n");
        exit(1);
    }
    v->previous = v->next = NULL;
    v->current = NOP;
    v->white = whitab;
    return v;
}

/*****************************************************************/
/*                                                               */
/*  sort white block in x order                                  */
/*                                                               */
/*****************************************************************/

struct whitablock **sort_x_white(headblock, numblock)
struct block_rect *headblock;
int  *numblock;
{
    int   i, j, stop;
    struct whitablock *tmpw, **blockx;

    *numblock = 0;
    for (tmpw = headblock->firstwhite; tmpw != NULL; tmpw = tmpw->next)
        (*numblock)++;

    blockx = (struct whitablock **) malloc((*numblock)*
                                    sizeof(struct whitablock *));
    if (blockx == NULL){
        printf("allocation error: in sort_x_white\n");
        exit(1);
    }

    for (tmpw = headblock->firstwhite, i =0; tmpw != NULL; tmpw = tmpw->next, i++)
        blockx[i] = tmpw;

    for (i = 0; i < *numblock ; i++){
        for (j = stop = 0; j < *numblock -1 -i; j++)
            if (blockx[j]->wbox.ll.x > blockx[j+1]->wbox.ll.x){
                tmpw = blockx[j];
                blockx[j] = blockx[j+1];
                blockx[j+1] = tmpw;
                stop = 1;
            }
        if (stop == 0) break;
    }

    return blockx;
}

/*****************************************************************/
/*                                                               */
/*  sort white block in x order                                  */
/*                                                               */
/*****************************************************************/

struct whitablock **sort_y_white(headblock, numblock)
struct block_rect *headblock;
int  *numblock;
{
    int   i, j, stop;
    struct whitablock *tmpw, **blocky;

    *numblock = 0;
    for (tmpw = headblock->firstwhite; tmpw != NULL; tmpw = tmpw->next)
        (*numblock)++;

    blocky = (struct whitablock **) malloc((*numblock)*
                                    sizeof(struct whitablock *));
    if (blocky == NULL){
        printf("allocation error: in sort_y_white\n");
        exit(1);
    }

    for (tmpw = headblock->firstwhite, i =0; tmpw != NULL; tmpw = tmpw->next, i++)
        blocky[i] = tmpw;

    for (i = 0; i < *numblock ; i++){
        for (j = stop = 0; j < *numblock -1 -i; j++)
            if (blocky[j]->wbox.ll.y > blocky[j+1]->wbox.ll.y){
                tmpw = blocky[j];
                blocky[j] = blocky[j+1];
                blocky[j+1] = tmpw;
                stop = 1;
            }
        if (stop == 0) break;
    }

    return blocky;
}

/*****************************************************************/
/*                                                               */
/*  first two top right_x                                        */
/*                                                               */
/*****************************************************************/

void first_x_twotop(wtable, top1, top2)
struct white_for_table *wtable;
int  *top1, *top2;
{
    struct white_for_table *tmpw, *tmp;
    int   num, stop, i, j;

    for (tmp = wtable, num = 0; tmp != NULL; tmp = tmp->next, num++)
    tmpw = (struct white_for_table **)malloc(num*
```

【図１２０】

```
for (tmp = wtable, num = 0; tmp != NULL; tmp = tmp->next, num++)
    tmp2[num] = tmp;

for ( i = 0; i < num ; i++){
    for (j = 0, stop = 0; j < num -1 -i; j++)
        if (tmp2[j]->white->mod_rightx> tmp2[j+1]->white->mod_rightx){
            tmp = tmp2[j];
            tmp2[j] = tmp2[j+1];
            tmp2[j+1] = tmp;
            stop = 1;
        }
    if (stop == 0) break;
}
*top2 = -1;
*top1 = tmp2[0]->white->mod_rightx;
for ( i = 1; i < num ; i++){
    if (tmp2[i]->white->mod_rightx != tmp2[0]->white->mod_rightx-OFFSET){
        if (tmp2[i]->white->mod_rightx > *top1)
            *top1 = tmp2[i]->white->mod_rightx;
    } else {
        *top2 = tmp2[i]->white->mod_rightx;
        break;
    }
}
if (*top2 == -1) *top2 = *top1;
}

/*****************************************************************************/
/*                                                                           */
/*      function:  first_top_lowxy                                           */
/*                                                                           */
/*****************************************************************************/

void first_top_lowxy(wtable, top1, top2)
struct white_for_table *wtable;
int  *top1, *top2;
{
    struct white_for_table **tmpw, **tmp;
    int  num, stop, i, j;

    for (tmp = wtable, num = 0; tmp != NULL; tmp = tmp->next, num++);
    tmpw = (struct white_for_table **)malloc(num*
                        sizeof(struct white_for_table*));
    for (tmp = wtable, num = 0; tmp != NULL; tmp = tmp->next, num++)
        tmpw[num] = tmp;

    for ( i = 0; i < num ; i++){
        for (j = 0, stop = 0; j < num -1 -i; j++)
            if (tmpw[j]->white->mod_lowxy> tmpw[j+1]->white->mod_lowxy){
                tmp = tmpw[j+1];
                tmpw[j+1] = tmpw[j];
                tmpw[j] = tmp;
                stop = 1;
            }
        if (stop == 0) break;
    }
    *top1 = -1;
    *top2 = tmpw[0]->white->mod_lowxy;
    for (i = 1; i < num ; i++){
        if (tmpw[i]->white->mod_lowxy == tmpw[0]->white->mod_lowxy-OFFSET){
            if (*top1 < tmpw[i]->white->mod_lowxy && tmpw[i]->white->mod_lowxy < *top2)
                *top1 = tmpw[i]->white->mod_lowxy;
        } else {
            *top2 = tmpw[i]->white->mod_lowxy;
            break;
        }
    }
}

/*****************************************************************************/
/*                                                                           */
/*      function:  combine two white                                         */
/*                                                                           */
/*****************************************************************************/

void combine_white(nblock, white1, white2)
struct block_list *nblock;
struct whiteblock *white1, *white2;
{
    struct whiteblock *tmpwhite;
    int   i, j, total;

    if (white1->numsubwhite == 0){
        total = white2->numsubwhite;
        if (white2->numsubwhite == 0) total++;
        if (white2->numsubwhite > new_white[0].total-1)
            new_white[0].total = white2->numsubwhite+1;
        for (i = 0; i <= white2->numsubwhite+1; i++)
            copy_white(&(white1->subwhite[i]), &(new_white[0]));
        free((struct whiteblock *)white1->subwhite);
        white1->subwhite = new_white[0].subwhite;
        white1->numsubwhite = total;
        white2->numsubwhite = 0;
    } else {
        total = white1->numsubwhite+white2->numsubwhite;
        if (white2->numsubwhite == 0) total++;
        tmpwhite = white1->subwhite;
        white1->subwhite = new_white[0].subwhite;
        for (i = 0; i <= white1->numsubwhite+1; i++)
            copy_white(&(white1->subwhite[i]), &(tmpwhite[i]));
        free((struct whiteblock *)tmpwhite);
        for (i = white1->numsubwhite; i <= total-2; i++)
            copy_white(&(white1->subwhite[i]), &(white2->subwhite[i-white1->numsubwhite]));
        white1->subwhite[total-1].mod_lowxy = white2->mod_lowxy;
        white1->numsubwhite = total;
        free((struct whiteblock *)white2->subwhite);
        free((struct whiteblock *)white2);
    }
}
```

【図１２１】

```
if (white2->mod_lowery > white1->mod_lowery)
  white1->mod_lowery = white2->mod_lowery;
if (white2->mod_leftx < white1->mod_leftx)
  white1->mod_leftx = white1->mod_leftx;
if (white2->mod_rightx > white1->mod_rightx)
  white1->mod_rightx = white2->mod_rightx;
white1->left_x = white1->mod_leftx;
white1->right_x = white1->mod_rightx;
white1->upper_y = white1->mod_uppery;
white1->lower_y = white1->mod_lowery;

if (nblock->firstwhite == white2){
  nblock->firstwhite = white2->next;
    if (nblock->firstwhite != NULL) nblock->firstwhite->previous = NULL;
    else nblock->currentwhite = NULL;
  } else if (nblock->currentwhite == white2){
      nblock->currentwhite = white2->previous;
      nblock->currentwhite->next = NULL;
    } else {
        white2->previous->next = white2->next;
        white2->next->previous = white2->previous;
      }
  free((struct whiteblock *) white2);
}

/*.....remove_vertical(nblock).....*/
/*.....Last Cable in Vertical direction.....*/
/*.....remove_vertical(nblock).....*/
void test_vertical(nblock)
struct block_rec *nblock;
{

  int    top1, top2, num, y, combine, stop, replace, max;
  struct whiteblock **a_white, *tmp;
  struct white_for_cable *head, *tmp1, *tmp2;

  top1 = nblock->upper_y;
  top2 = nblock->lower_y;
  replace = NIL; stop = NORMAL;
/*
printf("get into vertical\n");
*/
/*
for (tmp = nblock->firstwhite; tmp != NULL; tmp = tmp->next)
    printf("id x1 : %d y1 = %d x2 = %d y2 = %d id\n", tmp->index,
                tmp->mod_leftx, tmp->mod_uppery, tmp->mod_rightx, tmp->mod_lowery);
printf("top1 = %d top2 = %d \n", top1, top2);
*/
  a_white = sort_y_white(nblock, &num);
/*
for (y = 0; y < num; y++)
  printf("id x1 = %d y1 = %d x2 = %d y2 = %d \n", a_white[y]->index,
          a_white[y]->left_x, a_white[y]->upper_y, a_white[y]->right_x,
          a_white[y]->lower_y);
*/
  head = new_cable(a_white[0]);
  for (y = 1; (y < num) y++){
    if (a_white[y]->mod_uppery < top2) {
/*
  printf("id x1 = %d y1 = %d x2 = %d y2 = %d \n", a_white[y]->index,
          a_white[y]->mod_leftx, a_white[y]->mod_uppery, a_white[y]->mod_rightx,
          a_white[y]->mod_lowery);
*/
      for (tmp = head; tmp != NULL; tmp = tmp->next){
        if (a_white[y]->mod_uppery > tmp->mod_lowery){
/* printf("in spot1\n"); */
          if ((tmp->previous == NULL)||
              (a_white[y]->mod_uppery > tmp->previous->mod_lowery)){
            tmp1 = new_cable(a_white[y]);
            tmp1->next = tmp;
            tmp1->previous = tmp->previous;
            if (tmp->previous != NULL)
              tmp->previous->next = tmp1;
            tmp->previous = tmp1;
            if (tmp == head) head = tmp1;
            replace = YES;
            break;
          } else if (a_white[y]->mod_leftx >=
                     tmp->previous->mod_rightx-OFFSET){
/* printf("in spot2\n"); */
            if (tmp->next == NULL){
              tmp2 = new_cable(a_white[y]);
              tmp->next = tmp2;
              tmp2->previous = tmp;
              replace = YES;
              break;
            } else continue;
          } else {
              combine = 1;
              if (tmp->white->mod_lowery >=
                  a_white[y]->mod_uppery-OFFSET){
/* printf("in spot3\n"); */
                if (tmp->previous != NULL) {
                  if (((tmp->previous->white->mod_lowery
                      >= tmp->white->mod_lowery-OFFSET)&&
                      (tmp->previous->white->mod_lowery
                      <= tmp->white->mod_lowery+OFFSET))||
                      ((tmp->previous->white->mod_uppery
                      >= tmp->white->mod_lowery-OFFSET)&&
                      (tmp->previous->white->mod_uppery
                      <= tmp->white->mod_lowery+OFFSET))){
                    combine = 0;
                  }
                }
                if ((combine == 1)&&
                    (tmp->white->mod_rightx >
                     a_white[y]->mod_leftx-OFFSET)&&
                    (tmp->white->mod_leftx <
                     a_white[y]->mod_rightx+OFFSET)){
                  combine = 0;
                }
                if (combine == 1) {
/* printf("in spot4\n"); */
```

【図122】

【図123】

```
/*
for ( v = 0; v < num; v++)
    printf("%d x1 = %d y1 = %d x2 = %d y2 = %d \n", a_white(v)->index,
        a_white(v)->mod_leftx, a_white(v)->mod_upperp, a_white(v)->mod_rightx,
        a_white(v)->mod_lowerp);
*/
    head = new_wtable(a_white[0]);

    for (v = 1; (v < num); v++){
        if !a_white(v)->mod_leftx < top2) {
/*
    printf("%d x1 = %d y1 = %d x2 = %d y2 = %d \n", a_white(v)->index,
        a_white(v)->left_x, a_white(v)->upper_y, a_white(v)->right_x,
        a_white(v)->lower_y);
*/
        for (tmpv = head; tmpv != NULL; tmpv = tmpv->next){
            if !a_white(v)->mod_lowery <= tmpv->mod_upperp-OFFSET) {
/* printf("in spot1\n"); */
                if ((tmpv->previous != NULL)&&
                    (tmpv->previous->white == a_white[v])
                    break;
                tmpv2 = new_wtable(a_white[v]);
                if (tmpv == head){ tmpv2->next = tmpv;
                    head->previous = tmpv2;
                    head = tmpv2;
                } else {
                    tmpv2->previous = tmpv->previous;
                    tmpv2->next = tmpv;
                    tmpv2->next->previous = tmpv2;
                    tmpv2->previous->next = tmpv2;
                }
                replace = NOT;
                break;
            } else if (a_white[v]->mod_upperp <=
                tmpv->white->mod_lowery-OFFSET){
/* printf("in spot2\n"); */
                if (tmpv->next == NULL){
                    tmpv2 = new_wtable(a_white[v]);
                    tmpv->next = tmpv2;
                    tmpv2->previous = tmpv;
                    replace = NOT;
                    break;
                } else {
                    combine = 1;
                    if (tmpv->white->mod_rightx <=
                                a_white[v]->mod_leftx-OFFSET){
/* printf("in spot3\n"); */
/* printf("in spot4\n"); */
                        if ((tmpv->previous != NULL) {
                            if (((tmpv->previous->white->mod_rightx
                                >= tmpv->white->mod_leftx-OFFSET)&&
                                (tmpv->previous->white->mod_rightx
                                <= tmpv->white->mod_rightx+OFFSET))||
                                ((tmpv->previous->white->mod_leftx
                                >= tmpv->white->mod_leftx-OFFSET)&&
                                (tmpv->previous->white->mod_leftx
                                <= tmpv->white->mod_rightx+OFFSET))){
                                combine = 0;
                            }
                        }
/* printf("combine = %d \n", combine); */
                        if ((combine == 1)&&(tmpv->next != NULL)){/* printf("in spot5\n"); */
                            if (((tmpv->next->white->mod_rightx
                                >= tmpv->white->mod_leftx-OFFSET)&&
                                (tmpv->next->white->mod_rightx
                                <= tmpv->white->mod_rightx+OFFSET))||
                                ((tmpv->next->white->mod_leftx
                                >= tmpv->white->mod_leftx-OFFSET)&&
                                (tmpv->next->white->mod_leftx
                                <= tmpv->white->mod_rightx+OFFSET))){
                                combine = 1;
                            } else {
                                combine = 0;
                            }
                        }
/* printf("combine = %d \n", combine); */
                        if (combine == 1){
                            if (((tmpv->next->white->mod_rightx
                                >= a_white[v]->mod_leftx-OFFSET)&&
                                (tmpv->next->white->mod_rightx
                                <= a_white[v]->mod_rightx+OFFSET))||
                                ((tmpv->next->white->mod_leftx
                                >= a_white[v]->mod_leftx-OFFSET)&&
                                (tmpv->next->white->mod_leftx
                                <= a_white[v]->mod_rightx+OFFSET))){
                                combine = 2;
                            }
                        }
/* printf("combine = %d \n", combine); */
                        if (combine == 2) {
                            tmpv->white = a_white[v];
                            if(a_white[v]->mod_leftx<topl-OFFSET)
                                a_white[v]->mod_leftx = topl;
                            tmpv->status = NOT;
                            replace = YES;
                            break;
                        } else {
                            if (tmpv->status == OLD){
                                combine_white(tmpv,tmpv->white,a_white[v]);
                                tmpv->status = NOT;
                                replace = YES;
                                break;
                            } else {
                                if (combine == 1) {
                                    tmpv->white = a_white[v];
                                    if(a_white[v]->mod_leftx<topl-OFFSET)
                                        a_white[v]->mod_leftx = topl;
                                    tmpv->status = NOT;
                                    replace = YES;
                                    for (tmpv = tmpv->next; tmpv != NULL;){
                                        if((tmpv->white->mod_lowery <=
                                            a_white[v]->mod_lowery-OFFSET)||
                                            (tmpv->white->mod_leftx >
                                            a_white[v]->mod_leftx-OFFSET)){
                                            combine_white(tmpv,
                                                a_white[v],tmpv->white);
                                            tmpv->previous->next = tmpv->next;
                                            if (tmpv->next != NULL)
                                                tmpv->next->previous =
                                                    tmpv->previous;
                                            tmpv2 = tmpv->next;
                                            free((struct white_for_table *)
                                                tmpv);
                                            tmpv = tmpv2;
                                        } else break;
                                    }
                                } else {
                                    step = STOP;
                                    break;
                                }
                            }
                        }
                    }
                }
            }
        }
    }
/* printf("combine = %d \n", combine); */
    if (combine == 1){/* printf("in spot6\n"); */
        if (((tmpv->next->white->mod_rightx
            >= a_white[v]->mod_leftx-OFFSET)&&
            (tmpv->next->white->mod_rightx
            <= tmpv->white->mod_rightx+OFFSET))||
            ((tmpv->next->white->mod_leftx
            >= a_white[v]->mod_leftx-OFFSET)&&
            (tmpv->next->white->mod_leftx
            <= tmpv->white->mod_rightx+OFFSET))){
            combine = 2;
        }
    }
/* printf("combine = %d \n", combine); */
    if (combine == 2){/* printf("in spot7\n"); */
        if (((tmpv->next->white->mod_rightx
            <= a_white[v]->mod_rightx)&&
            (tmpv->next->white->mod_leftx
            >= a_white[v]->mod_leftx))){
            printf("tmpv - %d x = %d y = %d x2 = %d y2 = %d \n", tmpv->white->index,
                tmpv->next->white->mod_rightx, tmpv->white->mod_upperp, tmpv->white->mod_rightx,
```

【図124】

```
    tmpv->white->mod_lowery);

    */
        tmpv->status = OLD;
        if (tmpv->white == tmp->next->white) {
            tmpv2 = tmpv->next;
            tmpv->next = tmpv2->next;
            if (tmpv2->next != NULL)
                tmpv2->next->previous = tmpv;
            free((struct white_list *) tmpv2);
        } else tmpv = tmpv->next;
    }
    tmpv->status = OLD;
/*
printf("tmpv = %d x1 = %d y1 = %d x2 = %d y2 = %d \n", tmpv->white->index,
    tmpv->white->mod_leftx, tmpv->white->mod_upperx, tmpv->white->mod_rightx,
    tmpv->white->mod_lowery);
printf("in qbox3\n");
*/
    first_x_twoloop( head, &topl, &top2);
/* printf("after search topl= %d top2 = %d \n",*topl, top2); */
    if (topl == top2 ) top2 = nblock->right_x;
    stop = NONSTOP; replace = NOP;
    }
  }
  max = head->white->mod_rightx;
  for ( tmpv = head; tmpv != NULL; tmpv = tmpv->next)
    if (tmpv->white->mod_rightx > max)
      max = tmpv->white->mod_rightx;
  for ( tmpv = head; tmpv != NULL; tmpv = tmpv->next)
    if (tmpv->white->mod_rightx < max)
      tmpv->white->mod_rightx = max;

  free((struct whiteblock *)a_valid);
for (tmp = nblock->firstwhite; tmp != NULL; tmp = tmp->next){
/* printf("%d x1 = %d y1 = %d x2 = %d \n", tmp->index,
    tmp->mod_leftx, tmp->mod_upperx, tmp->mod_rightx, tmp->mod_lowery);
*/
/*
    if (tmp->numsubwhite != 0){
      for (v = 0; v < tmp->numsubwhite; v++)
        printf(" v = %d index = %d x1 = %d y1 = %d x2 = %d y2 = %d \n",
          v, tmp->subwhite[v].index, tmp->subwhite[v].mod_leftx,
          tmp->subwhite[v].mod_upperx, tmp->subwhite[v].mod_rightx,
          tmp->subwhite[v].mod_lowery);
    } else
      printf("%d x1 = %d y1 = %d x2 = %d y2 = %d \n", tmp->index,
        tmp->mod_leftx, tmp->mod_upperx, tmp->mod_rightx, tmp->mod_lowery);
*/
}
/*
printf("get out of horizontal\n");
*/
}

/*****************************************************************************/
/********** table test                                                       **********/
/*****************************************************************************/

int t(table)(nblock)
struct block_rect *nblock;
{
    int original_white, white1, white2;

    struct whiteblock *tmp;

    original_white = nblock->current_white->index;

/* first time test */
    test_horizontal(nblock);
    white1 = 0;
    for (tmp=nblock->firstwhite; tmp != NULL; tmp = tmp->next, white1++);
/* printf("original_white = %d white1 = %d \n",original_white, white1); */
    white_valid(a_valid);
    for (tmp = nblock->firstwhite; tmp != NULL; tmp = tmp->next, white2++);
/* printf("original_white = %d white2 = %d \n",original_white, white2);*/

    if ((white2 < 4)||(white2 < original_white/2)) return 0;
    else if ((white2 == original_white)||
             ((white1 == white2)&&(white1 == 0.8*original_white)))
                return 1;

/* second time test */
    test_horizontal(nblock);
    white1 = 0;
    for (tmp=nblock->firstwhite; tmp != NULL; tmp = tmp->next, white1++);
/* printf("original_white = %d white1 = %d \n",original_white, white1); */
    if ((white1 < 4)||(white1 < original_white/2)) return 0;
    else if ((white1 == white2)&&(white1 >= 0.8*original_white))
            return 1;
    test_vertical(nblock);
    white2 = 0;
    for (tmp = nblock->firstwhite; tmp != NULL; tmp = tmp->next, white2++);
/* printf("original_white = %d white2 = %d \n",original_white, white2); */
    if ((white2 == white1)&&(white2 >= 0.8*original_white))
      return 1;
    else
      return 0;
}
```

【図125】

```
/*****************************************************************************/
/*****************************************************************************/
/****************** File: block_util.c                          ******************/
/****************** Author: Shin-Yuan Wang                      ******************/
/****************** Date : 1-28-92                              ******************/
/****************** Copyright 1992 Canon Information Systems    ******************/
/*****************************************************************************/
/*****************************************************************************/


/*****************************************************************************/
/******** test if covered range overlap                                ********/
/*****************************************************************************/

int if_overlap(a1, a2, b1, b2)
int a1, a2, b1, b2;
{
     if ((a2 < b1)||(b2 < a1)) return 0;
     return 1;
}


/*****************************************************************************/
/********  minimum of two integers                                     ********/
/*****************************************************************************/

int min(a, b)
int a, b;
{
    if (a <= b) return a;
    else return b;
}


/*****************************************************************************/
/********  ratio of overlappint                                        ********/
/*****************************************************************************/

float cover_range(a1, a2, b1,  b2)
int a1, a2, b1, b2;
{
     float range = 0.0;
     int   x1, x2;

     if ((a2 < b1) || (b2 < a1)) return range;
     else {
           x1 = max(a1, b1);
           x2 = min(a2, b2);
          }

     range = ((float)(x2-x1+1)/(float)(a2-a1+1));
     return range;
}
```

【図131】

```
/*****************************************************************************/
/*****************************************************************************/
/*********************    Filename: blockline.h                   ************/
/**********************    Author: Shin-Yuan Wang                 ************/
/*********************    Date:   1-15-93                         ************/
/*********************** Copyright 1991 Canon Information Systems ************/
/*****************************************************************************/
/*****************************************************************************/

#define EMPTY       0
#define OVERLAP     1
#define NEXT        2
#define SURROUNDIN  3
#define SURROUNDOUT 4

#define TH          1.7
#define WIDTHSTD    1.2
#define CAPLENGTH   0

#define DOT_BOTTOM  0.8
#define DOT_WIDTH   1
#define LINEOVERLAP 0.5

#define LEFTGAP     0x10
#define RIGHTGAP    0x01
#define NOISE_NO    3

typedef int fillcondition;

extern    void xm_draw_box();
extern    void convert_headblock();
extern    void transfer_block();
extern    void remove_textblock();
extern    int  max();
extern    int  min();
extern    float cover_range();
extern    struct blockline **new_linepointer();
extern    void sort_linepointer();
```

【図１２７】

```
/*****************************************************************************/
/*********************   Filename: block_main.h                    ***********/
/*********************   Author: Shin-Yuan Wang                    ***********/
/*********************   Date:   1-15-93                           ***********/
/*****************  Copyright 1992 Canon Information Systems      ************/
/*****************************************************************************/
/*****************************************************************************/

#define TESTTIME 20
#define TEXT_BLOCK 0x11;
#define LINEPICTURE_BLOCK 0x21
#define HPICTURE_BLOCK    0x31
#define TABLE_BLOCK       0x41
#define LINE_BLOCK        0x51
#define FRAME_BLOCK       0x61
#define UNKNOWN_BLOCK     0x71

#define TEXT_COLOR        0
#define TABLETEXT_COLOR   1
#define LINE_COLOR        2
#define PICTURE_COLOR     3
#define HPICTURE_COLOR    4
#define UNKNOWN_COLOR     5
#define TABLE_COLOR       6
#define FRAME_COLOR       7
#define LINEMAP_COLOR     8

enum {SP_READ = 0x01,
      SP_REDUCE = 0x02,
      SP_COLOR = 0x04,
      SEP_BLOCK = 0x08};


struct block_image
{
  int width;
  int height;
  unsigned char** pixel;
};

int xoff[TESTTIME][2] = {{0, 1}, {1, 2}, {0, 1}, {1, 2}, {0, 1},
                         {1, 2}, {0, 0}, {0, 0}, {1, 1}, {1, 1},
                         {2, 2}, {2, 2}, {2, 0}, {2, 1}, {1, 0},
                         {2, 1}, {0, 1}, {0, 1}, {1, 2}, {1, 2}};

int yoff[TESTTIME][2] = {{0, 0}, {0, 0}, {1, 1}, {1, 1}, {2, 2},
                         {1, 2}, {0, 1}, {1, 2}, {0, 1}, {1, 2},
                         {0, 1}, {0, 2}, {0, 1}, {0, 1}, {2, 2},
                         {1, 2}, {1, 2}, {0, 1}, {1, 2}, {0, 1}};

#define  TOTALCOLOR  7

char *color_label[]={"                    Text          ",
                     "              Text in the table    ",
                     "                    Picture       ",
                     "            Line Drawing Graphics ",
                     "  Solid Line (Vertical or Horizontal)",
                     "                    Table         ",
                     "                    Frame         "};


int color_code[]={TEXT_COLOR, TABLETEXT_COLOR, HPICTURE_COLOR, PICTURE_COLOR,
                  LINE_COLOR, TABLE_COLOR, FRAME_COLOR};
```

【図１３２】

```
/*****************************************************************************/
/*****************************************************************************/
/*********************   Filename: blocksection.h                  ***********/
/*********************   Author: Shin-Yuan Wang                    ***********/
/*********************   Date:   1-15-93                           ***********/
/*****************  Copyright 1992 Canon Information Systems      ************/
/*****************************************************************************/
/*****************************************************************************/

#define GAP 16
#define OVERLAP 0.8
#define HOVERLAP 0.5
#define LEVELOVERLAP 0.0
#define LEFTSIDE     0x80
#define LEFTALIGN    0x40
#define LEFTTOPLONGER 0x20
#define LEFTBOTLONGER 0x10
#define RIGHTSIDE    0x0f
#define RIGHTALIGN   0x08
#define RIGHTTOPLONGER 0x04
#define RIGHTBOTLONGER 0x02
#define TEXTPIXEL    0x08

#define NOISE_COUNT   5

extern int line_inside();
extern int max();
extern int min();
extern int if_overlap();
extern void ma_draw_box();
extern    float cover_range();
void   exit();


struct segment_gap
{  int column_no;
  int *begin_column;
  int *end_column;
};
```

【図128】

```
/*****************************************************************/struct block_rect
/*                                                               */{
/*                 File: blockgeneral.h                          */   int   x, y;
/*                 Author: Kuh-Kwan Wang                         */   int   right_x, left_x;
/*                 Date : 1-4-92                                 */   int   upper_y, lower_y;
/*                 Copyright 1992 Canon Information Systems      */   int   width, length;
/*                                                               */   int   index;
/*****************************************************************/   int   numsubblock;
                                                                      int   group;
#define UNKNOWN 0x0000                                                int   n_group;                  /* the number of 'i' */
#define TEXT 0x01                                                     int   ch_point;                 /* text, picture, table, or straight line */
#define PICTURE 0x02                                                  int   att;
#define RPICTURE 0x04                                                 int   cover;
#define TABLE 0x08                                                    int   is_sorted;
#define HSOLID 0x10                                                   int   *pairright;
#define VSOLID 0x20                                                   int   *pairleft;
#define FRAME 0x40                                                    float avg_height;
#define LINE 0x80                                                     float density;
#define DOUBT 0x100                                                   struct block_rect *next;
#define UP 0x200                                                      struct block_rect *previous;
#define TEXT2 0x401                                                   struct block_rect *firstsubblock;
#define CAPTION 0x800                                                 struct block_rect *firstnextsubblock;
#define NOTDEF7 0x1000                                                struct block_rect *currentnonsubblock;
#define NOTDEF8 0x2000                                                struct block_rect *currentsubblock;
#define FRONT 130,014.0/71.0                                          struct whiteblock *firstwhite, *currentwhite;
#define MAXPOP 100                                                    };
#define MAIN_GRAPH 4
#define NORMALE_SPACE 8.6                                             struct linablock
#define SCALE_RATIO 1                                                 {
#define STANDARDFONT 12.0*MAXPOW/(72.0*SCALE_RATIO)                     struct block_rect *first_block;
#define CAPMAX      0x40                                                struct block_rect *current_block;
                                                                        struct linablock  *firstsublina;
                                                                        struct linablock  *currentsublina;
struct h_outline                                                        struct linablock  *firstnonsubline;
{                                                                       struct linablock  *currentnonsubline;
  int x;                                                                struct linablock  *previous;
  struct h_outline *next;                                               struct linablock  *next;
};
                                                                        int   sidegap;
                                                                        int   group;
struct data_outline                                                     int   upper_y, lower_y, left_x, right_x;
{                                                                       int   width, length;
  int   total_left;                                                     int   att;
  int   total_right;                                                    int   nsubblock;
  struct h_outline *headleft;                                           int   index;
  struct h_outline *headright;                                          struct block_rect *under_caption;
  struct h_outline *current_left;                                       int   i_caption, r_caption;
  struct h_outline *current_right;                                    };
};

                                                                      struct sameline
struct whiteblock                                                     {
{                                                                       struct sameline *previous;
  int  x, y;                                                            struct sameline *next;
  int  right_x, left_x;                                                 struct linablock *current;
  int  upper_y, lower_y;                                                struct sameline  *i_connection;
  int  nx_right, nx_left;                                               struct sameline  *r_connection;
  int  nupper_y, nx_lower_y;                                            struct sameline  *i_connection;
  int  width, length;                                                   int              index;
  int  line_with_block;                                               };
  int  nsubwhite;
  int  index;                                                         struct sectionblock
  struct whiteblock *previous, *next;                                 {
  struct whiteblock *subwhite;                                          struct linablock   *firstlinablock;
  int  **pairright;                                                     struct linablock   *currentlinablock;
  int  **pairleft;                                                      struct sectionblock *firstsubsection;
};                                                                      struct sectionblock *currentsubsection;
                                                                        struct sectionblock *firstnextsection;
                                                                        struct sectionblock *currentnextsection;
```

【図130】

```
/*************************************************************************/
/*********************    Filename: blockrect.h          ****************/
/*********************    Author: Shin-Ywan Wang         ****************/
/*********************    Date:   1-15-92                ****************/
/****************** Copyright 1992 Canon Information Systems ************/
/*************************************************************************/
/*************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#define ALLDIRECT        8
#define ONE              0x01
#define ZERO             0

#define LEFT_ONE         0x11
#define RIGHT_ONE        0x21
#define LEFTRIGHT_ONE    0x31
#define LEFT_ZERO        0x10
#define RIGHT_ZERO       0x20
#define RIGHTLEFT_ZERO   0x30

#define SOLIDTRED        9.0
#define SOLIDDENSITY     0.9
#define SINDEGREE1       0.017452406
#define HDOT             8
#define TEXT_SCALE       2.0
#define WIDTH_RATIO      4
#define GAPY             1

#define COVER            1
#define COVERED          -1
#define ORIGINAL         2

#define CENTER           0
#define NW               1
#define WEST             2
#define SW               3
#define SOUTH            4
#define SE               5
#define EAST             6
#define NE               7
#define NORTH            8


#define SKIP  0
#define LEFT  1
#define RIGHT 2
#define RL    3

typedef int recording;

extern int if_overlap();
extern if_table();
extern float cover_range();

struct loc_point
{ int x;
  int y;} around_center[9]={(0,0), (-1, -1), (-1, 0),
                            (-1, 1), (0, 1), (1, 1),
                            (1, 0), (1, -1), (0, -1)};

int search_map[9][8]={(0,0,0,0,0,0,0, 0),
                      (7, NE, NORTH, NW, WEST, SW, SOUTH, SE),   /* for NORTH_WEST */
                      (6, NW, WEST, SW, SOUTH, SE, EAST, 0),     /* for WEST */
                      (7, NW, WEST, SW, SOUTH, SE, EAST, NE),    /* for SOUTH_WEST */
                      (6, SW, SOUTH, SE, EAST, NE, NORTH, 0),    /* for SOUTH */
                      (7, SW, SOUTH, SE, EAST, NE, NORTH, NW),   /* for SOUTH_EAST */
                      (6, SE, EAST, NE, NORTH, NW, WEST, 0),     /* for EAST */
                      (7, SE, EAST, NE, NORTH, NW, WEST, SW),    /* for NORTH_EAST */
                      (6, NE, NORTH, NW, WEST, SW, SOUTH, 0)};   /* for NORTH */

recording  map_inf[9][8]={
         {SKIP, SKIP, SKIP, SKIP, SKIP, SKIP, SKIP, SKIP},
         {SKIP, LEFT, LEFT, LEFT, LEFT, RL, RL, RL},  /*  NORTH_WEST */
         {SKIP, LEFT, RIGHT, RIGHT, RIGHT, RIGHT,RIGHT , SKIP}, /*  WEST */
         {SKIP, SKIP, RIGHT, RIGHT, RIGHT, RIGHT, RIGHT, RL},  /*  SOUTH_WEST */
         {SKIP, RIGHT, RIGHT, RIGHT, RIGHT, RL, RL, SKIP},     /*  SOUTH */
         {SKIP, RIGHT, RIGHT, RIGHT, RIGHT, RL, RL, RL},  /*  SOUTH_EAST */
         {SKIP, RIGHT, LEFT, LEFT, LEFT, LEFT, LEFT, SKIP},    /*  EAST */
         {SKIP, SKIP, LEFT, LEFT, LEFT, LEFT, LEFT, RL}, ./*  NORTH_EAST */
         {SKIP, LEFT, LEFT, LEFT, LEFT, RL, RL, RL}}; /*  NORTH */
```

【図135】

```
/*****************************************************************************/
/*********************   Filename: table.h                     *****************/
/*********************   Author: Shin-Ywan Wang                ******************/
/*********************   Date:   1-25-93                       ******************/
/*****************  Copyright 1992 Canon Information Systems   ******************/
/*****************************************************************************/
/*****************************************************************************/

#include "blockgeneral.h"

#define NONE 0
#define OFFSET  3
#define OFFSET2 4
#define NEW 1
#define OLD 2
#define NONSTOP 0
#define STOP 1
#define TABLE_STARTSIZE 3

struct white_for_table
{
  int    status;
  struct whiteblock *white;
  struct white_for_table *previous, *next;
};
```

【図136】

```
/*****************************************************************************/
/*********************   Filename: images.h                    *****************/
/*********************   Author: Shin-Ywan Wang                ******************/
/*********************   Date:   1-13-93                       ******************/
/*****************  Copyright 1992 Canon Information Systems   ******************/
/*****************************************************************************/
/*****************************************************************************/

#include<stdio.h>
#include<stdlib.h>

struct images
{
  int sizeR;
  int sizeC;
  int dpi;
  unsigned char **pixel;
};
```

【図142】

```
/*
 * Filename: autoinput.h
 * Header file for autoinput.c
 *
 *****************************************************************
 */


/* Function Definitions for mymalloc.c */
#ifdef CPP
/* Definition for C++ files */
extern "C" {
  void auto_initialize();
  int auto_input_int();
  void auto_terminate();
}
#else
/* Definition for C files */
void auto_initialize();
int auto_input_int();
void auto_terminate();
#endif
```

【図145】

```
/*-------------------------------------------------------------*/
void Zerovpp_terminate(nzlist)
     struct NonZeroVppList *nzlist; /* The list */
{
  free_ivector(nzlist->list);   /* Deallocate memory */
  nzlist->allocated = 0;        /* Set so won't be accidently used */
  nzlist->size = 0;
}

/*-------------------------------------------------------------*/
void Zerovpp_add(nzlist, a, b)
     struct NonZeroVppList *nzlist; /* The list */
     int a,b;                       /* Start and stop of the Zerovpp */
{
  if (nzlist->size+2 > nzlist->allocated) {
    printf("NonZeroVppList FULL!!!\n");
    exit(1);
  }
  nzlist->list[(nzlist->size)++] = a;
  nzlist->list[(nzlist->size)++] = b;
}
```

【図244】

```
    colors[x].flags = DoRed | DoGreen | DoBlue;

    colors[x].red = the_color;
    colors[x].green = the_color;
    colors[x].blue = the_color;
    the_color += 300;
  }
  XQueryColors(d,def_colormap,colors,ncolors);

  my_cmap = XCreateColormap(d,DefaultRootWindow(d),DefaultVisual(d,scr),
                            AllocAll);
  XStoreColors(d,my_cmap,colors,ncolors);
}
*/
```

【図138】

```
CC=     cc
CPP=    CC
COMPILE_PROFILE = -p
COMPILE_STYLE = -O -g
COMPILE_VISIBLE = -DDEBUG_OUTPUT=VISIBLE

LDLIBS= -lm
LDFLAGS= -L/usr/lib/DCLB4 -L/usr/lib/ANSI/libm.1 -lXm -lXt -lX11 -lmalloc -lPW
CPPFLAGS= -I/usr/include/X11R4 -DCPP ${CCOMPILE_STYLE} ${COMPILE_VISIBLE}
CFLAGS= -I/usr/include/X11R4 ${COMPILE_STYLE} ${COMPILE_VISIBLE} ${COMPILE_PROFILE}
OBJECTS_multilevel = pragma.o linprocess.o 3line_logic.o outline.o combine.o
line.o zerovpp.o charlist.o cimages.o mymalloc.o xs.o xt.o util.o recognize_init.o test_multi.o
line.o cutdash.o scalsprint.o this_file.o autoinput.o
OBJECTS_test = test.o charlist.o mymalloc.o

all: multilevel test

multilevel: ${OBJECTS_multilevel}
        ${CPP} -o multilevel ${OBJECTS_multilevel} ${LDLIBS} ${LDLIBSX}

test:: ${OBJECTS_test} ${LDLIBS} ${LDLIBSX}
        ${CC} -o test ${OBJECTS_test} ${LDLIBS} ${LDLIBSX}

3line_logic.o: 3line_logic.c
        ${CPP} -c ${CPPFLAGS} 3line_logic.c

autoinput.o: autoinput.c
        ${CC} -c ${CFLAGS} autoinput.c

combine.o: combine.c
        ${CPP} -c ${CPPFLAGS} combine.c

cimages.o: cimages.c
        ${CPP} -c ${CPPFLAGS} cimages.c

cutdash.o: cutdash.c
        ${CPP} -c ${CPPFLAGS} cutdash.c

linprocess.o: linprocess.c
        ${CPP} -c ${CPPFLAGS} linprocess.c

multilevel.o: multilevel.c
        ${CPP} -c ${CPPFLAGS} multilevel.c

multilevel2.o: multilevel2.c
        ${CPP} -c ${CPPFLAGS} multilevel2.c

mymalloc.o: mymalloc.c
        ${CC} -c ${CFLAGS} mymalloc.c

optimum.o: optimum.s
        ${CC} -s ${CFLAGS} optimum.s

outline.o: outline.c
        ${CPP} -c ${CPPFLAGS} outline.c

pragment.o: pragment.c
        ${CPP} -c ${CPPFLAGS} pragment.c

recognize.o: recognize.c
        ${CC} -c ${CFLAGS} recognize.c

segment.o: segment.c
        ${CPP} -c ${CPPFLAGS} segment.c

level3a1.o: level3a1.c
        ${CPP} -c ${CPPFLAGS} level3a1.c

level3a2.o: level3a2.c
        ${CPP} -c ${CPPFLAGS} level3a2.c

scalsprint.o: scalsprint.c
        ${CPP} -c ${CPPFLAGS} scalsprint.c

test.o: test.c
        ${CC} -c ${CFLAGS} test.c

test_multi2.o: test_multi2.c
        ${CPP} -c ${CPPFLAGS} test_multi2.c

this_file.o: this_file.c
        ${CC} -c ${CFLAGS} this_file.c

util.o: util.c
        ${CC} -c ${CFLAGS} util.c

xs.o: xs.c
        ${CC} -c ${CFLAGS} xs.c

xt.o: xt.c
        ${CC} -c -DCPP ${CFLAGS} xt.c

zerovpp.o: zerovpp.c
        ${CPP} -D ${CPPFLAGS} zerovpp.c

charlist.o: charlist.h mymalloc.h
combine.o: charlist.h cimages.h
cimages.o: charlist.h cimages.h xs.h
level3.o: mymalloc.h
linprocess: charlist.h cimages.h
multilevel.o: charlist.h xs.h mymalloc.h
multilevel2.o: charlist.h cimages.h
pragment.o: charlist.h xs.h mymalloc.h
segment.o: charlist.h cimages.h
zerovpp.o: charlist.h cimages.h

rm_visible:
        rm linprocess.o multilevel2.o pragment.o recognize.o scalsprint.o
```

【図139】

```
//
// Filename: 3line_logic.c
// This routine contains optimized 3line logic routines
//
//////////////////////////////////////////////////////////////////////
//
#include"images.h"

//.....
// 2x1 OR logic
void line_OR_2(oldline, changes newline)
{
  int older, olde;          // Size of old image
  int newr, newe;           // Size of new image
  int i,j,a;
  unsigned char *oc1, *oc2, *ncr
  unsigned char *or, *nr;   // The old & new column pointers

  older = oldline.sizeb;
  olde = oldline.sizec;
  newr = (older-1)/2;       // Should be ==
  newe = olde;
  newline.realloconte(newr-1, newe);
  a = older - newr-2;       // How much is partially covered in the last row
  if (a == 2)
    newr++;                 // It's evenly aligned

  or = oldline.pixel;
  nr = newline.pixel;

  for (i=0; i<newr; i++)    // Do the middle
  {
    oc1 = *or++;
    oc2 = *(or++);
    nc = *(nr++);
    for (j=0; j<newe; j++)
      *(nc++) = *(oc1++) | *(oc2++);
  }

  // Do the ends
  nc = *nr;
  if (a == 1)
  {
    oc1 = *or++;
    for (j=0; j<newe; j++)
      *(nc++) = *(oc1++);
  }
}

//.....
// 2x1 OR logic
void line_OR_3(oldline, changes newline)
{
  int older, olde;          // Size of old image
  int newr, newe;           // Size of new image
  int i,j,a;
  unsigned char *or1, *oc1, *oc2, *nc; // The old & new column pointers
  unsigned char **or, **nr; // The old & new row pointers

  older = oldline.sizeb;
  olde = oldline.sizec;
  newr = (older-1)/2;       // Should be ==
  newe = olde;
  newline.realloconte(newr-1, newe);
  a = older - newr-2;       // How much is partially covered in the last row

  if (a == 2)
    newr++;                 // It's evenly aligned

  or = oldline.pixel;
  nr = newline.pixel;

  for (i=0; i<newr; i++)    // Do the middle
  {
    oc1 = *or++;
    oc2 = *(or++);
    nc = *(nr++);
    for (j=0; j<newe; j++)
      *(nc++) = *(oc1++) | *(oc2++);
  }

  // Do the ends
  nc = *nr;
  if (a == 1)
  {
    oc1 = *or++;
    for (j=0; j<newe; j++)
      *(nc++) = *(oc1++) | *(oc2++);
  }
}

//.....
// 2x1 OR logic
void line_OR_4(oldline, changes newline)
{
  int older, olde;          // Size of old image
  int newr, newe;           // Size of new image
  int i,j,a;
  unsigned char *or1, *oc1, *oc2, *nc; // The old & new column pointers
  unsigned char **or, **nr; // The old & new row pointers

  older = oldline.sizeb;
  olde = oldline.sizec;
  newr = (older-1)/2;       // Should be ==
  newe = olde;
  newline.realloconte(newr-1, newe);
  a = older - newr-2;       // How much is partially covered in the last row

  if (a == 2)
    newr++;                 // It's evenly aligned

  or = oldline.pixel;
  nr = newline.pixel;

  for (i=0; i<newr; i++)    // Do the middle
  {
    oc1 = *or++;
    oc2 = *(or++);
    nc = *(nr++);
    for (j=0; j<newe; j++)
      *(nc++) = *(oc1++) | *(oc2++);
  }

  // Do the ends
  nc = *nr;
  if (a == 1)
  {
    oc1 = *or++;
    for (j=0; j<newe; j++)
      *(nc++) = *(oc1++) | *(oc2++);
  }
}

//.....
// 1x2 AND 2x1 OR logic
void line_AOR_12_2(oldline, changes newline)
{
  int older, olde;          // Size of old image
  int newr, newe;           // Size of new image
  int i,j,a,b;
  unsigned char *oc1, *oc2, *nc;  // The old & new column pointers
  unsigned char **or, **nr; // The old & new row pointers
  register unsigned char temp1, temp2; // Temporary holding variable
  older = oldline.sizeb;
  olde = oldline.sizec;
  newr = (older-1)/2;       // Should be ==
  newe = (olde-1)/2;        // Should be ==
  newline.realloconte(newr-1, newe-1);

  a = older - newr-2;       // How much is partially covered in the last row
  if (a == 2)
    newr++;                 // It's evenly aligned
  b = older - newr-2;       // How much is partially covered in the last column
  if (b == 2)
  {
    newe++;
    b = 0;                  // == 0 for optimization of ends below
  }
  or = oldline.pixel;
  nr = newline.pixel;

  for (i=0; i<newr; i++)    // Do the middle
  {
    oc1 = *(or++);
```

【図140】

```
    oc2 = *dst++;
    nc = *(nr++);
    for (j=0; j<new; j++)
    {
      temp1 = *(oc1++);
      temp2 = *(oc2++);
      *(nc++) = (temp1 & *(oc1++))  (temp2 & *(oc2++));
    }
    if (b)
      *(nc) = (*(oc1)  *(oc1));
  }

  // Do the bottom line (if a == 1)
  nc = *nr;
  if (a == 1)                       // a == 1  it's one row
  {
    oc1 = *or;
    for (j=0; j<new; j++)
    {
      temp1 = *(oc1++);
      *(nc++) = temp1 & *(oc1++);
    }
    if (b)                          // Do the right end
      *(nc) = *(oc1);
  }
}

// ..........................................................
// 1x2 AND 1x1 OR Logic
void slims_and_or_1x2(oldims, dimage& newims)
{
  int old, older;               // Size of old image
  int newr, newc;               // Size of new image
  int i,j,a,b;
  unsigned char *oc1, *oc2, *nr; // The old & new column pointers
  unsigned char **or, **nr;     // The old & new row pointers
  register unsigned char temp1, temp2;

  older = oldims.sized;
  oldc = oldims.sizec;
  newr = (older-1)/2;           // Should be ==
  newc = (oldc-1)/2;            // Should be ==
  newlims.resize(newr+1, newc+1);

  a = older - newr*2;           // How much is partially covered in the last row
  if (a == 1)
    newr++;                     // It's evenly aligned
  b = oldc - newc*2;            // How much is partially covered in the last column
  if (b == 2)
  {
    newc++;
    b = 0;
  }
  or = oldims.pixel;
  nr = newims.pixel;

  for (i=0; i<newr; i++)        // Do the middle
  {
    oc1 = *(or++);
    oc2 = *(or++);
    nc = *(nr++);
    for (j=0; j<newc; j++)
    {
      temp1 = *(oc1++);
      temp2 = *(oc2++);
      temp1 = (temp1 & *(oc1++))  (temp2 & *(oc2++));
      *(nc++) = (temp1 & *(oc1++))  (temp2 & *(oc2++));
    }
    if (b)                      // Do the right end
      *(nc) = *(oc1)  *(oc2);
  }

  // Do the bottom line (if a == 1 or 2)
  nc = *nr;
  if (a == 1)                       // a == 1  it's one row
  {
    oc1 = *or;
    for (j=0; j<newc; j++)
    {
      temp1 = *(oc1++);
      *(nc++) = temp1 & *(oc1++);
    }
    if (b)                          // Do the right end
      *(nc) = *(oc1);
  }
  else {                            // a == 2  it's two rows
    oc1 = *(or++);
    oc2 = *or;
    for (j=0; j<newc; j++)
    {
      temp1 = *(oc1++);
      temp2 = *(oc2++);
      *(nc++) = (temp1 & *(oc1++))  (temp2 & *(oc2++));
    }
    if (b)                          // Do the right end
      *(nc) = *(oc1)  *(oc2);
  }
}
```

【図141】

```
/*
 * Filename: autoinput.h
 * This file contains the code to input from a filename
 *
 *.....
 */

#include <stdio.h>
#include "autoinput.h"

/* Global variables */

int auto_eof_detected,      /* 1= 0 if eof has been detected */
FILE *auto_fp;              /* The file pointer */
int save_to_file;           /* Saving the input to a file? */
int save_count;             /* Number of inputs before newline */
char filename[80];          /* The filename */

/*.....*/
void auto_initialise()
{
  int not_exit = 1;

  auto_eof_detected = 0;

  while(not_exit)
  {
    printf("Input filename for auto-input? ");
    scanf("%s", filename);

    auto_fp = fopen(filename, "r");

    if (auto_fp == NULL)      /* Read open failed */
    {
      auto_fp = fopen(filename, "w");
      if (auto_fp == NULL)    /* Write open failed */
      {
        printf("Cannot open file %s.\n", filename);
      } else {                /* Opened new file */
        printf("%s opened as a new file.\n", filename);
        printf("Enter 1 at the next prompt.\n");
        save_to_file = 1;
        auto_eof_detected = 1;
        not_exit = 0;
      }
    } else {                  /* Read open succeeded */
      printf("Opening existing file %s\n",filename);
      not_exit = 0;
    }
  }

  printf("Do you wish to save the auto-input into this file? (1 - yes)? ");
  scanf("%d", &save_to_file);
  auto_input_int();
  printf("Afterwards change the last '1' to a '0' to turn off appending to the file\n");
  save_count = 0;
}


int auto_input_int()
{
  int i;

  if (auto_eof_detected)
  {
    printf("\n(Manual input (-9 = newline, -999 = exit)) ? ");
    scanf("%d", &i);
    if (i == -999)
    {
      fclose(auto_fp);
      exit(0);
    }
    if (save_to_file == 1)
    {
      if (i == -9)                /* Print new lines */
      {
        fprintf(auto_fp, "\n"); /* Print the new lines */
        fprintf(auto_fp, "\n");
        fflush(auto_fp);
        save_count = 0;
        return auto_input_int(); /* And input again */
      }
      fprintf(auto_fp, "%d ", i);
      fflush(auto_fp);
      if (++save_count > 3)
      {
        save_count = 0;
        fprintf(auto_fp, "\n");
      }
      return i;
    }

    if (fscanf(auto_fp, "%d", &i) == EOF)
    {
      printf("\n**** END OF FILE DETECTED ****\n");
      if (save_to_file == 1)
      {
        fclose(auto_fp);
        auto_fp = fopen(filename, "a");
        if (auto_fp == NULL)
        {
          printf("***** WARNING: CANNOT APPEND TO FILE: %s *****\n", filename);
          save_to_file = 0;
        } else
          fprintf(auto_fp, "\n\n");

        auto_eof_detected = 1;
        return auto_input_int();
      }
      printf("\n\n");
      fclose(auto_fp);
    }
    return i;
  }
```

【図143】

```
/*
 * Filename: charlist.c
 * This file contains Charlist and RasterGyplist utility routines
 *
 *
 */

/*-------------------------------------------------------------------*/
/* ..... Routines to handle Character lists                          */
/*-------------------------------------------------------------------*/
/* By Christopher Sharlock */

#include"charlist.h"
#include"xmalloc.h"

/*-------------------------------------------------------------------*/
/* Initialize and allocate a Charactelist (set up curve pointers) */
/* Call this routine once only! */
void Charlist_initialize(clist, clist_max, vector_height)
    struct Charlist *clist;
    int clist_max;          /* The MAX number of characters */
    int vector_height;      /* Height of the line (to allocate curves) */
{
    int i;                  /* A looping var */

    /* Allocate memory for characters */
    clist->chr = (struct Character *) Xalloc((unsigned) clist_max
                                    * sizeof(struct Character));
    if (!clist->chr) {
        printf("Memory Allocation Error in Charlist_initialize()\n");
        exit(1);
    }
    clist->size = 0;            /* Number of characters */
    clist->allocated = clist_max; /* Size Allocated */
    clist->first = 0;           /* No first character */
    clist->vector_height = vector_height;

    /* Allocate memory for curves and set pointers */
    /* NOTE: To optimize, hardcode the array into the code */
    for (i=0; i<clist_max; i++) {
        clist->chr[i].curve = (vector(vector_height));
        clist->chr[i].curve = (vector(vector_height));
    }
}

/*-------------------------------------------------------------------*/
/* Zero a previously allocated Characterlist */
void Charlist_zero(clist)
    struct Charlist *clist;
{
    clist->size = 0;            /* Number of characters */
    clist->first = 0;           /* No first character */
}

/*-------------------------------------------------------------------*/
/* Terminate deallocate the Characterlist */
void Charlist_terminate(clist)
    struct Charlist *clist;
{
    int i;

    for (i=clist->allocated-1; i>= 0; i--) { /* Free curves */
        free_(vector(clist->chr[i].curve));
        free_(vector(clist->chr[i].curve));
    }
}

free((struct Character *) clist->chr);
clist->size = 0;
clist->allocated = 0;
clist->first = 0;
}

/*-------------------------------------------------------------------*/
/* Place a new character in the Charlist. */
/* Calculates the character beyond the charlist ( chr[size]).
/* Also calculates slope and inserts it into the list */
void Charlist_place_new_char(clist)
    struct Charlist *clist;
{
    struct Character *newchar;   /* Pointer to the new character adding */

    newchar = &(clist->chr[clist->size]); /* Set newchar */

    /* Calculate the position of the new character */
    Charlist_recalculate_loc(newchar);

    /* Check for full list */          /* Increment the size of the list */
    (clist->size)++;
    if (clist->size >= clist->allocated) /* If too big.. */
    {
        printf("Error in Charlist_place_new_char(): Character list is FULL!\n");
        exit(1);
    }

    /* Place new in the list character */
    Charlist_locate_character(clist, newchar);
}

/*-------------------------------------------------------------------*/
/* This routine repositions a character in the charlist.  It removes
 * the character, recalculates loc and inserts it in the list accordingly
 */
void Charlist_replace_char(clist, ch)
    struct Charlist *clist;
    struct Character *ch;
{
    /* Recalculate the position of the character */
    Charlist_recalculate_loc(ch);

    if (clist->size < 2)        /* If it's the only item in the list */
        return;

    /* Remove the character from the list */
    if (ch->prev == 0) {        /* If 1st on list */
        clist->first = clist->first->next;
        clist->first->prev = 0;
    } else {
        ch->prev->next = ch->next;
        if (ch->next != 0)
            ch->next->prev = ch->prev;
    }

    /* Place the character in the list */
    Charlist_locate_character(clist, ch);
}

/*-------------------------------------------------------------------*/
/* Insert a character after doing a cut */
```

【図144】

```
/* This routine will insert the character beyond the charlist i.chr[size]).
 * .loc is recalculated for both ch_after and the new character; however
 * .loc is NOT USED TO PLACE THE CHARACTER.  It will
 * place the new character after ch_after!  After finished, it is an
 * extremely good idea to resort the Worklist!
 */
void Charlist_insert_after_cc(clist, ch_after)
  struct Charlist *clist;
  struct Character *ch_after;  /* Place the new character after this one */
{
  struct Character *newchar;  /* Pointer to the new character adding */

  newchar = &(clist->charlist->xinsel); /* Get newchar */

  /* Calculate the position of the new character */
  Charlist_recalculate_loc(newchar);
  Charlist_recalculate_loc(ch_after);

  /* Check for full list */
  (clist->size)++;
  if (clist->size >= clist->allocated) /* If too big... */
    printf("Error in Charlist_place_new_char(): Character list is FULL!!!\n");
    exit(1);
  }

  newchar->prev = ch_after;
  newchar->next = ch_after->next;
  ch_after->next = newchar;
  if (newchar->next != 0)
    newchar->next->prev = newchar;
}

/*.....................................................................*/
/* This routine will remove a character from the character list
 * Note: The memory is NOT reclaimed!!!  So don't rely on this too much.
 * Also, prev and next pointers of chars are NOT altered :)
 */
void Charlist_remove_char(clist, ch)
    struct Charlist *clist;
    struct Character *ch;
{
  /* Remove the character from the list */
  if (ch->prev == 0) /* If loc on list */
    clist->first = clist->first->next;
  else clist->first->prev = 0;
} else {
  ch->prev->next = ch->next;
  if (ch->next != 0)
    ch->next->prev = ch->prev;
}
}

/*.....................................................................*/
/* This routine will recalculate the position of a character */
void Charlist_recalculate_loc(ch) /* helper to the position */
  struct Character *ch; /* The character */
{
  int i,loc;
  /* Calculate the position of the new character */
  loc = 0;
  for (i=ch->top; i<ch->bot; i++) /* Motion = average position... */
    loc += ch->curve2[i] + ch->curve3[i]; /* ... of the curves */
  loc = loc * 4 / (ch->bot - ch->top);
  ch->loc = loc;
                    /* Set it */
}

/*.....................................................................*/
/* This routine will place a character that is not in the charlist!
 * Into the charlist. */
void Charlist_locate_character(clist, ch)
    struct Charlist *clist;
    struct Character *ch;  /* The character */
{
  int loc;
  struct Character *p;   /* Used for searching for location */
  struct Character *last_p;  /* The character before p */

  /* Is the list empty? */
  if (clist->size == 1)
  {
    ch->next = 0;
    ch->prev = 0;
    clist->first = ch
    return;
  }

  /* Possible optimization: Don't start from start of list...
       start from where at */
  loc = ch->loc;      /* Retrieve position */

  /* Is it at the beginning of the list */
  p = clist->first;
  if (loc < p->loc)
  {
    p->prev = ch;
    ch->next = clist->first;
    clist->first = ch;
    return;
  }

  while (p->loc <= loc)
  {
    last_p = p;
    p = p->next;
    if (p == 0)
      break;
  }
  ch->prev = last_p;
  last_p->next = ch;
  if (p!=0)
    p->prev = ch;
}

/*.....................................................................*/
/*.....................  Zero Vxp List Stuff  .........................*/
/*.....................................................................*/
void Vxpapp_InitializeVxpList(vxplist, size)
  struct MonsterVxpList *vxplist; /* The list */
  int size;                       /* The number of entries (2x allocated) */
{
  /* Allocate memory for list */
  size = size*2;
  vxplist->list = (vector(size); /* Allocate memory */
  vxplist->allocated = size;     /* Set how allocated */
  vxplist->size = 0;             /* Set to zero elements */
}
```

【図146】

```
/*
 * Filename: charlist.h
 * Header file for charlist.c
 *
 */

/*****************************************************************/
/* ENUM: chartype: Different types of characters */
typedef enum {
  CHARACTER_UNRECOGNISED,
  CHARACTER_ENTIRE_LINE,
  CHARACTER_LEFT_OUT,
  CHARACTER_NOT_OUT_ENOUGH,
  CHARACTER_COMBINED
chartype;

typedef enum {
  TRIED_RECOGNITION,
  FAILED_RECOGNITION,
  UNRECOGNISED
charRecognition;                    /* Recognition hasn't been tried on it */

/*****************************************************************/
/* Struct Character:
 * Contains information about an outlined character in a page
 */
struct Character {
  chartype type;                    /* The type of character */
  charRecognition rec;              /* If it's recognized */
  int level_cut_at;                 /* The level the character was cut at
                                       (for goodness) */
  int loc;                          /* The location of the character */
  int top;                          /* The start of the character (top)
                                       (inclusive) */
  int bot;                          /* The bottom of the character (bottom)
                                       (exclusive) */
  int left;                         /* The left most pixel in this character */
  int right;                        /* The right most pixel in this character */
  int *curve1;                      /* The left side of the character(inclusive)
                                       (-1 , not there) */
  int *curve2;                      /* The right side of the character (exclusive)
                                       (-1 , not there) */
  struct Character *next;           /* Ptr to next character in the list
                                       (0 if last character) */
  struct Character *prev;           /* Ptr to previous character in the list
                                       (0 if first character) */
};

/*****************************************************************/
/* Struct Charlist:
 * Contains information about the list of characters */
struct Charlist {
  struct Character *chr;            /* Array: The characters */
  int size;                         /* Number of characters in the chr[] */
  int allocated;                    /* Number of characters allocated */
  int vector_height;                /* The height of the character curves */
  struct Character *first;          /* The first character in linked list */
};

/*****************************************************************/
/* Struct MonkeroVpplist:
 * Contains information about a list for man-zero VPP's: */

struct MonZeroVpplist {
  int* list;                        /* Array: in a stuff, 2b+1 * step */
  int allocated;                    /* Memory allocated (1,2,4...) */
  int size;                         /* How many used (1,2,4...) */
};

/*****************************************************************/
/*                                                               */
/*                        Define Routines                        */
/*                                                               */
/*****************************************************************/

#ifdef CPP

/* Define C++ Routines */
extern "C" {
  void Charlist_initialise(struct Charlist *clist, int clist_max,
                           int vector_height);
  void Charlist_zero(struct Charlist *clist);
  void Charlist_terminate(struct Charlist *clist);
  void Charlist_place_new_char(struct Charlist *clist);
  void Charlist_replace_char(struct Charlist *clist, struct Character *ch);
  void Charlist_insert_after_cut(struct Charlist *clist,
                                 struct Character *ch_after);
  void Charlist_remove_char(struct Charlist *clist, struct Character *ch);
  void Charlist_recalculate_loc(struct Character *ch);
  void Charlist_locate_character(struct Charlist *clist, struct Character *ch);
  void Zerovpp_initialise(struct MonZeroVpplist *mzlist, int size);
  void Zerovpp_coordinate(struct MonZeroVpplist *mzlist);
  void Zerovpp_add(struct MonZeroVpplist *mzlist, int a, int b);
}

#else

/* Define C routines */
void Charlist_initialise();
void Charlist_zero();
void Charlist_terminate();
void Charlist_place_new_char();
void Charlist_replace_char();
void Charlist_insert_after_cut();
void Charlist_remove_char();
void Charlist_recalculate_loc();
void Charlist_locate_character();
void Zerovpp_initialise();
void Zerovpp_coordinate();
void Zerovpp_add();

#endif
```

【図147】

```
//
//  Filename:  cimages.cpp
//  This file contains a vast array image utility functions, for a cimage
//
//////////////////////////////////////////////////////////////////////////

// undefine this to make images.C run invisibally (except for error)
//#define PRINT_DEBUG_INFO

#include"cimages.h"
#include"x.h"

//.....
//... CLASS: cimage   (an array of chars)
//.....

cimage::cimage()
{
  sizeX = 0;
  sizeC = 0;
  allocated_sizeX = 0;
  allocated_sizeC = 0;
  cpi = 0;
}

cimage::cimage(int x, int c)
{
  sizeX = 0;                    // no problems won't be caused when
  sizeC = 0;                    //   allocate called deallocate
  allocated_sizeX = 0;
  allocated_sizeC = 0;
  cpi = 0;
  allocate(x,c);
};

cimage::cimage(cimage& img)
{
  allocate(sizeX, sizeC); // Allocate memory for new cimage (loads sizes)
  for (int i=0; i<sizeX; i++)     // copy old image to new cimage
    for (int j=0; j<sizeC; j++)
      pixel[i][j] = img.pixel[i][j];
};

//////////////////////////////////////////////////////////////////////////
// Delete the cimage
cimage::~cimage()
{
  deallocate();
};

//////////////////////////////////////////////////////////////////////////
// Allocate memory for an cimage from memory
void cimage::allocate(int x, int c)
{
  deallocate();         // Remove current cimage
  if (x < 0 || c < 0)
  {
    cerr << "Illegal cimage size: " << x << " by " << c << "\n";
    exit(1);
  }
  sizeX = x;
  sizeC = c;
  if ((x != 0) && (c != 0))
  {
    pixel = (unsigned char **)malloc((unsigned int) sizeX * sizeof(unsigned char*));
    // Allocate more mem
    if (pixel[0] == NULL)
    {
      cerr << "Error: Memory Allocation Error (out of memory)\n";
      exit(1);
    }
    for (int i = 0; i<sizeX; i++)
    {
      pixel[i] = (unsigned char *)malloc((unsigned int) sizeC * sizeof(unsigned char));
      // Allocate more mem
      if (pixel[i] == NULL)
      {
        cerr << "Error: Memory Allocation Error (out of memory)\n";
        exit(1);
      }
    }
    allocated_sizeX = sizeX;
    allocated_sizeC = sizeC;
  }
}

//////////////////////////////////////////////////////////////////////////
// Deallocate memory for an cimage from memory
void cimage::deallocate()
{
  if ((allocated_sizeX != 0) && (allocated_sizeC != 0))
  {
    for (int i = 0; i < allocated_sizeX; i++)
      free((char *) pixel[i]);
    if (pixel != 0)
      free((char **) pixel);        // Zero out the sizes
    sizeX = 0;
    sizeC = 0;
    allocated_sizeX = 0;
    allocated_sizeC = 0;
  }
}

//////////////////////////////////////////////////////////////////////////
// This allows allocation of memory from outside the class.  It will
// Allocate memory if its smaller.  If not, it will just adjust the size
void cimage::reallocate(int R, int C)
{
  int new_R, new_C;
  if ((R > allocated_sizeX) || (C > allocated_sizeC))
  {
    new_R = allocated_sizeX;
    new_C = allocated_sizeC;
    if (R > new_R)
      new_R = R;
    if (C > new_C)
      new_C = C;
    allocate(new_R,new_C);      // Re-allocate the image
    printf("..... NOTE: IMAGESIZE REALLOCATED TO (%d, %d)\n", new_R, new_C);
  }
  sizeX = R;
  sizeC = C;
}

//////////////////////////////////////////////////////////////////////////
// Make a cimage with a single value: value = value, X and Y are the size
void cimage::fill(char value, int R, int C)
```

【図148】

```
{
  allocate(R,C);
  for (int i=0; i<size; i++)
    for (int j=0; j<size; j++)
      pixel[i][j] = value;
}

////////////////////////////////////////////////////////////////////
// Copy a section of a image into another.  Image a reference image
// Range is [R1..R2) [C1..C2)
void image::clip(image& image, int R1, int R2, int C1, int C2)
{
  int r,c,t;

  if (R2 < R1)                  // Swap R1 & R2
    { t=R1; R1=R2; R2=t; }
  if (C2 < C1)                  // Swap C1 & C2
    { t=C1; C1=C2; C2=t; }
  if (R1 < 0)  R1 = 0;          // Check limits
  if (R2 < 0)  R2 = 0;
  if (C1 < 0)  C1 = 0;
  if (C2 < 0)  C2 = 0;
  if (R1 > image.size1())  R1 = image.size1();
  if (R2 > image.size1())  R2 = image.size1();
  if (C1 > image.size2())  C1 = image.size2();
  if (C2 > image.size2())  C2 = image.size2();

  allocate(R2-R1, C2-C1);

  dpl = image.dpl;
  for (r=0; r<R2-R1; r++)
    for (c=0; c<C2-C1; c++)
      pixel[r][c] = image.pixel[r+R1][c+C1];
}

////////////////////////////////////////////////////////////////////
// Clipspace reduce the size of an image by removing the blank
// space around it.  This adjusts the image size.
void image::clipspace()
{
  int R1, R2, C1, C2;  // The starting and stopping rows of new image
  int i, j;
  struct image temp;   // Temporary image

  R1 = size1;          // Initialize to extremes
  R2 = 0;
  C1 = size2;
  C2 = 0;

  for (i=0; i<size1; i++)       // Compute dimensions of new image
    for (j=0; j<size2; j++)
      if (pixel[i][j] != 0)
        {
          if (i < R1) R1 = i;   // Left edge?
          if (i > R2) R2 = i;   // Right edge?
          if (j < C1) C1 = j;   // Top edge?
          if (j > C2) C2 = j;   // Left edge?
        }

  if (R1 > R2) R1 = R2 = 0;     // Check for no pixel found
  if (C1 > C2) C1 = C2 = 0;

  temp.allocate(R2-R1+1, C2-C1+1);      // New image is this size

  for (i=R1; i<=R2; i++)
    for (j=C1; j<=C2; j++)
      temp.pixel[i-R1][j-C1] = pixel[i][j];     // Copy clipped image

  allocate(temp.size1, temp.size2);     // Copy temp image to current image
  for (i=0; i<size1; i++)
    for (j=0; j<size2; j++)
      pixel[i][j] = temp.pixel[i][j];
}

////////////////////////////////////////////////////////////////////
// Read in part of a Tiff file
void image::readtiff(char *filename)
{
  FILE *tagfile;                // Input file
  long int tagfile_position;    // The cur file position (in bytes)
  long int width;               // Width of picture
  long int length;              // Length of picture
  long int black;               // Value of Black pixel
  long int bps;                 // Bytes per strip
  long int raster_offset;       // Start of image data
  long int numtags, tag, type, len, value;  // Tag values
  long int i,j;

  if ((tagfile = fopen(filename, "rb")) == NULL)
  {
    cout << "\nError in opening " << filename << "\n";
    exit(1);
  }
  tagfile_position = 0;

  // Read in header and make sure it's Intel
  i = getbytes(tagfile,tagfile_position);
  if (i != 0x4949)
  {
    cout << "... Error: TIFF file not in Intel Format\n";
    exit(1);
  }

  // Read in Magic number
  i = getbytes(tagfile,tagfile_position);
  if (i != 0x002A)
  {
    cout << " Bad Magic Number for TIFF file!!!\n";
    exit(1);
  }

  // Find First File directory
  j = getbytes(tagfile,tagfile_position);
  while(tagfile_position < j);
  {
    i = getbyte(tagfile,tagfile_position);
  }

  // Tags in Directory
  numtags = getbytes(tagfile,tagfile_position);

  while (--numtags >= 0) {
    tag = getbytes(tagfile, tagfile_position);
    type = getbytes(tagfile, tagfile_position);
    len = getbytes(tagfile, tagfile_position);
    value = getbytes(tagfile, tagfile_position);
    if ((len != 1) && ((tag & 32768) == 0))
    {
      cout << "Error: Tag Length != 1\n";
```

【図149】

```
    exit(1);
  }

  switch (tag)
  {
    case 0x00FE:          // NewSubFileType
      if (value != 0)
      {
        cout << "Can't deal with Non-Standard NewSubFileTypes\n";
        exit(1);
      }
      break;
    case 0x00FF:          // OldSubFileType
      if (value != 1)
      {
        cout << "SubFile Type not full resolution\n";
        exit(1);
      }
      break;
    case 0x0100:          // Width
      width = value;
      break;
    case 0x0101:          // Length
      length = value;
      break;
    case 0x0102:          // Bits per sample
      if (value != 1)
      {
        cout << "; Error: Bits/Sample must be 1";
        exit(1);
      }
      break;
    case 0x0103:          // Compression
      if (value != 1)
      {
        cout << "Compression Used( Error)\n";
        exit(1);
      }
      break;
    case 0x0106:          // Black is
      black = value;
      break;
    case 0x0107:          // Threshholding tag
      if ((value < 1) || (value > 1))
      {
        cout << "Threshholding Tag = Illegal value\n";
        exit(1);
      }
      break;
    case 0x0111:          // Offset to Raster Data
      raster_offset = value;
      break;
    case 0x0117:
      bps = value;
      break;
    case 0x011a:
      break;
    case 0x011b:
      break;
    case 0x0128:
      break;
    default:
      if ((value & 32768) == 0) {
        cout << "0000 Minimum An unsupported tag appears in the TIF file\n";
      }
      break;
  } // switch
} // while

while(tagfile_position < raster_offset)
{
  i = getbyte(tagfile,tagfile_position);
}

//-------------------   READ IN FILE   ------------------------------
allocate((int) length, (int) width);

int v;

for (i = 0; i<length; i++)
{
  for (j = 0; j<width; j++)
  {
    if ((j%8) == 0)
    {
      v = getc(tagfile);
      tagfile_position++;
      if (v == EOF)
      {
        cerr << "\n Error: Premature End of File found in tiff file\n";
        exit(1);
      }
      if ((v & 128) == black*128)
        pixel[i][j] = 0;
      else
        pixel[i][j] = 1;
      v = v << 1;
    }
  }
}
fclose(tagfile);
}

////////////////////////////////////////////////////////////////////////
// Read in a byte.  File position is updated
// eoferror = 1 to error on EOF, 0 not
int cimage::getbyte(FILE *f, long int file_position, int eoferror)
{
  int ch;
  ch = getc(f);
  if ((ch == EOF) && (eoferror == 1))
  {
    cerr << "\n Error: Premature End of File (in getbyte())\n";
    exit(1);
  }
  file_position++;
  return ch;
}

////////////////////////////////////////////////////////////////////////
// Read in an integer from a file.  (LSB, MSB)
long int cimage::get2bytes(FILE *f, long int fp)
{
  int c1, c2;
  c1 = getbyte(f, fp, 1);
  c2 = getbyte(f, fp, 1);
  return c1 + c2*256;
}

////////////////////////////////////////////////////////////////////////
// Read in a long integer
long int cimage::get4bytes(FILE *f, long int fp)
{
```

【図150】

```
  long int c1, c2;
  c1 = getlbytes(f, fp);
  c2 = getlbytes(f, fp);
  return c1 + c2*65536;
}


////////////////////////////////////////////////////////////////////
// Plot image using Flash Graphics at (xloc, yloc) with color
void cimage::plot(int x, int y)
{
  int px,py;            // Pixel coords on the screen (upper left)
  char color;

  for (py=0; py<sizeR; py++)
    for (px=0; px<sizeC; px++)
      {
        color = pixel[py][px];
        if (color & 1)
          xs_plotxy(x+px, y+py); // Draw it
      }
};


////////////////////////////////////////////////////////////////////
// Plot image with a box around it (with color 1!)
void cimage::plotbox(int x, int y)
{
  int px,py;

  for (px=0; px<sizeC+2; px++) {        // Draw horizontal lines
    if ((x+px+y)%2 == 0)
      xs_plotxy(x+px, y);
    if ((x+px+y+sizeR+1)%2 == 0)
      xs_plotxy(x+px, y+sizeR+1);
  }
  for (py=1; py<sizeR+1; py++) {
    if ((x+y+py)%2 == 0)
      xs_plotxy(x, y+py);
    if ((x+sizeC+1+y+py)%2 == 0)
      xs_plotxy(x+sizeC+1, y+py);
  }
  plot(x+1, y+1);       // Plot the image
}
```

【図151】

```
//
//  Filename:   cimages.h
//  This is the header file for cimages.C
//
////////////////////////////////////////////////////////////////////
//
#define MEM_DEBUG 1
#include<iostream.h>
#include<stdio.h>
#include<stdlib.h>
#include<math.h>

class cimage;           // Forward reference

//*****************************************************************
//*** CLASS: cimage
//*****************************************************************
class cimage
{
public:
  int           sizeR;          // Number of rows (first index of array)
  int           sizeC;          // Number of cols. (second index of array)
  int           allocated_sizeR; // Amount malloced
  int           allocated_sizeC; // Amount malloced
  int           dpi;            // Dots per inch of the image (0 if unknown)
  unsigned char **pixel;        // The binary image (2d array)

  cimage();                     // Initialize
  ~cimage();                    // Destroy
  cimage(int r, int c);         // Initialize with a size
  cimage(cimage& temp);

  void reallocate(int R, int C); // Will allocate memory for an image
  void fill(char value, int R, int C); // Fill image with a constant value
  void clip(cimage& image, int R1, int R2, int C1, int C2);
  void clipspace();             // Clips zero space surrounding image
  void readtiff(char *filename);  // Read in a tiff image
  void plot(int X, int y);      // Plot Binary image
  void plotbox(int x, int y); // Plot with a box around it

private:
  void allocate(int r, int c);        // Allocate memory
  void deallocate();                  // Deallocate memory
  int getbyte(FILE *f, long int& file_position, int eoferror);
  long int get2bytes(FILE *f, long int& fp);
  long int get4bytes(FILE *f, long int& fp);
};
```

【図167】

```
  extern void cut_dash(cimage& line, struct Charlist* clist);
  cut_dash(*line_to_process), clist);
#endif

#ifdef COMBINING
  charlist_combine(*line_to_process), clist);
#endif

#ifdef PRINT_FINAL_CHARLIST
  //////// Print the characterlist
  xs_clr_win();
  print_charlist(*line_to_process), clist, 10, 10);
  xs_flush();
#endif

#ifdef DISPLAY
  xs_flush();
#endif
  printf("Displaying Final Segmentation.  Enter 0 to continue) ");
  int dummy;
  dummy = auto_input_int();

  terrorpp_terminate(&masternslist);     // *** DELETE ***
}
```

【図220】

```
/*
 * Filename: autoinput.h
 * Header file for autoinput.c
 *
 ----------------------------------------------------------------
 */


/* Function Definitions for mymalloc.c */
#ifdef CPP
/* Definition for C++ files */
extern "C" {
  void auto_initialize();
  int auto_input_int();
  void auto_terminate();
}
#else
/* Definition for C files */
void auto_initialize();
int auto_input_int();
void auto_terminate();
#endif
```

【図235】

```
/*
 * Filename: mymalloc.h
 * Header file for mymalloc.c
 *
 ----------------------------------------------------------------
 */


/* Function Definitions for mymalloc.c */
#ifdef CPP
/* Definition for C++ files */
extern "C" {
  int *ivector(int n);
  void free_ivector(int* v);
  float *fvector(int n);
  void free_fvector(float* v);
  int *ivector_ranged(int m, int n);
  void free_ivector_ranged(int* v, int m);
  int **imatrix(int r, int c);
  void free_imatrix(int** m, int r);
  char **cmatrix(int r, int c);
  void free_cmatrix(char** m, int r);
}
#else
/* Definition for C files /*
/* The header file for mallocs */
int *ivector();
void free_ivector();
int *fvector();
void free_fvector();
int *ivector_ranged();
void free_ivector_ranged();
int **imatrix();
void free_imatrix();
char **cmatrix();
void free_cmatrix();
#endif
```

【図152】

```
//
// Filename: combine.C
// Routine for combining incorrectly separated characters
//
////////////////////////////////////////////////////////////////////////

#include "charline.h"
#include "image.h"
#include "xs.h"

#define VERY_SMALL -32000       // Just for debugging!!  DELETE EVENTUALLY

#define VERY_SMALL -32000       // A very small integer (for image size)

#define FRAC_VALID_DIST 3.0     // How far apart breaks must be (in XX pixels
                                // count) to be overlapping.  Multiply this
                                // by the smaller dot to get range.

#define DOT_TO_BODY_HEIGHT_RATIO 2 // How much longer the body of the i must be
                                // than the dot (body = X x dot)

#define PERCENT_TYPED_RATIO 0.75 // The ratio of the dot the line should
#define PERCENT_TYPED_MIN 0.25   // Minimum ratio of dot width / line width
#define PERCENT_TYPED_MAX 0.5    // Maximum ratio of dot width / line width
#define PERCENT_TYPED_SLOPE 0.1  // Inverse of how much bottom slot for computing the
                                // slope of how much percent algo' to see

#define FRACT_PERCENT_AVERAGE 0.25 // Multiply this by width of dot to see
                                // how far the characters must overlap

#define FONT_DOT_VIEW_MIN 0.5   // Minimum ratio of dot width / line width
#define FONT_DOT_VIEW_MAX 1.25  // Maximum ratio of dot width / line width
#define FONT_DOT_HEIGHT_MIN 0.45 // Minimum ratio of dot height / line height
#define FONT_DOT_HEIGHT_MAX 0.75 // Maximum ratio of dot height / line height
#define FONT_MIN_SLOPE 0.33      // Minimum inverse slope of the line (as
                                // inverse of maximum slope of line)

// Define a Combinelist which is a list of 4 characters (with information
// stored about them) that are looked at to see if we need to combine them.
struct Combinelist {
  struct Character *c1;         // The first character in the list
  struct Character *c2;         // The second character in the list
  struct Character *c3;         // ... (and so on) (0 = none)
  struct Character *c4;
  struct Character *next;       // The next character to look in
  int h12;                      // The height difference between 1 & 2
  int h23;                      // (-1 = undefined, 0 = chars overlaps
  int h34;                      // (1 means first higher, 2 means 2nd higher)
  int d12;                      // The horizontal distance between 1 & 2
  int d23;                      // (defined from top of 1 to bottom of other)
  int d34;                      // (VERY_SMALL = undefined)
};

// Define external variables and routines (for displaying and debugging)
extern int curnum_i;
extern int curnum_j;
extern print_character(image& line, struct Character *chr, int x, int y);
extern print_charlist(image& line, struct Charlist *clist, int x, int y);

////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////
//////      Character debugging routines                          //////
////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////
void test_print(
    image& line,        // Image of the line
    struct Charlist *clist,     // The Charlists
    struct Combinelist *comb)   // The list
{
  int x,y;

  x = y = 10;

  xs_clr_win();
  if (comb->c1 != 0)
  {
    print_character(line, comb->c1, x, y);
    x += 10 - comb->c1->left + comb->c1->right;
  }

  if (comb->c2 != 0)
  {
    print_character(line, comb->c2, x, y);
    x += 10 - comb->c2->left + comb->c2->right;
  }

  if (comb->c3 != 0)
  {
    print_character(line, comb->c3, x, y);
    x += 10 - comb->c3->left + comb->c3->right;
  }

  if (comb->c4 != 0)
  {
    print_character(line, comb->c4, x, y);
    x += 10 - comb->c4->left + comb->c4->right;
  }

  print_charlist(line, clist, 10, 20+clist->vector_height);
  xs_flush();
}


////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////
//////                                                            //////
//////                  Character combining routines              //////
//////                                                            //////
////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////
// This routines returns if two characters are above each other.
// Returns 0 for overlaping or character does not exist, 1 for 1st higher,
// 2 for 2nd higher.  It also updates h12.
int comb_height_calc(
    struct Character *c1,       // The first character
    int *h12)                   // The height (updated)
{
  if (*h12 != -1)               // If it's been calculated,
    return (*h12);              // don't recalculate it

  if (c2 == 0)                  // If character doesn't exist
  {
    *h12 = 0;                   // Return no gap exists.
    return 0;
  }

  if (c1->bot <= c2->top)       // If bottom of c1 is higher than top of c2
  {
    *h12 = 1;                   // Return the gap
    return 1;
  }

  if (c1->bot >= c2->top)       // If bottom of c2 is higher than top of c1
  {
    *h12 = 2;                   // Return the gap
    return 2;
  }

  *h12 = 0;                     // Return no gap exists
  return 0;
}
```

[図153]

```
//////////////////////////////////////////////////////////////////
// This routine is called from comb_dist() with a likly i or j
// characters.  This determines if close enough to be combined.
// This will use the right side of the character to compute closeness
int comb_dist_j(
  struct Character *c1,         // The first character
  struct Character *c2,         // The second character
  int hi2,                      // Which character is higher
  int *di2,                     // The last value
  int width_small)              // The width of the smaller character
{
  int toppos, botpos;           // Positions of the top and bottoms
  int i;
  float f;

  if (hi2 == 1)
  ////// Case when c1 is higher than c2
  {
    // Find bottom pos of c1
    if ((c1->bot - c1->top) < 4)
      botpos = c1->loc;         // If character very small, use .loc
    else
    {
      botpos = 0;
      for (i=c1->bot - 4; i < c1->bot; i++)
        botpos += c1->curve2[i] + c1->curve2[i];
    }

    // Find top pos of c2
    if ((c2->bot - c2->top) < 4)
      toppos = c2->loc;         // If character very small, use .loc
    else
    {
      toppos = 0;
      for (i=c2->top; i < c2->top + 4; i++)
        toppos += c2->curve2[i] + c2->curve2[i];
    }

    // Compute position and exit
    *di2 = toppos - botpos;

    f = ((float) *di2) / ((float) width_small);
    if (f < FRAC_VALID_DIST)
      return 1;
    else
      return 0;
  }

  ////// c2 is higher than c1
  // Find top pos of c1
  if ((c1->bot - c1->top) < 4)
    toppos = c1->loc;   // If character very small, use .loc
  else
  {
    toppos = 0;
    for (i=c1->top; i < c1->top + 4; i++)
      toppos += c1->curve2[i] + c1->curve2[i];
  }

  // Find bottom pos of c2
  if ((c2->bot - c2->top) < 4)
    botpos = c2->loc;   // If character very small, use .loc
  else
  {
    botpos = 0;
    for (i=c2->bot - 4; i < c2->bot; i++)
      botpos += c2->curve2[i] + c2->curve2[i];
  }

  // Compute position and exit
  *di2 = botpos - toppos;

  f = ((float) *di2) / ((float) width_small);
  if (f < FRAC_VALID_DIST)
    return 1;
  else
    return 0;
}


//////////////////////////////////////////////////////////////////
// This routine is called from comb_dist() with a pair of characters
//that probably isn't an i or j (.,!=)
// characters.  This determines if close enough to be combined.
int comb_dist_not(
  struct Character *c1,         // The first character
  struct Character *c2,         // The second character
  int hi2,                      // Which character is higher
  int *di2,                     // The last value
  int width_small)              // The width of the smaller character
{
  int toppos, botpos;           // Positions of the top and bottoms
  int i;
  float f;

  if (hi2 == 1)
  ////// Case when c1 is higher than c2
  {
    // Find bottom pos of c1
    if ((c1->bot - c1->top) < 4)
      botpos = c1->loc;         // If character very small, use .loc
    else
    {
      botpos = 0;
      for (i=c1->bot - 4; i < c1->bot; i++)
        botpos += c1->curve1[i] + c1->curve2[i];
    }

    // Find top pos of c2
    if ((c2->bot - c2->top) < 4)
      toppos = c2->loc;         // If character very small, use .loc
    else
    {
      toppos = 0;
      for (i=c2->top; i < c2->top + 4; i++)
        toppos += c2->curve1[i] + c2->curve2[i];
    }

    // Compute position and exit
    *di2 = toppos - botpos;

    f = ((float) *di2) / ((float) width_small);
    if (f < FRAC_VALID_DIST)
      return 1;
    else
      return 0;
  }

  ////// c2 is higher than c1
  // Find top pos of c1
  if ((c1->bot - c1->top) < 4)
    toppos = c1->loc;   // If character very small, use .loc
```

[図154]

```
  else
  {
    toppos = 0;
    for (i=c1->top; i < c1->top + 4; i++)
      toppos += c1->curve1[i] + c1->curve2[i];
  }

  // Find bottom pos of c2
  if ((c2->bot - c2->top) < 4)
    botpos = c2->loc;   // If character very small, use .loc
  else
  {
    botpos = 0;
    for (i=c2->bot - 4; i < c2->bot; i++)
      botpos += c2->curve1[i] + c2->curve2[i];
  }

  // Compute position and exit
  *dl2 = botpos - toppos;

  f = ((float) *dl2) / ((float) width_small);
  if (f < FRAC_VALID_DIST)
    return 1;
  else
    return 0;
}


////////////////////////////////////////////////////////////////
// This routines returns how close two non-height-overlapping characters
// are (hl2 must be valid!).  Computes characters from top of lower one
// to bottom of higher one.  Returns 0 = Bad, 1 = Good.  Also updates dl2
// Position is generated to be relative to dot height
int comb_dist(
  struct Character *c1,         // The first character
  struct Character *c2,         // The second character
  int hl2,                      // Which character is higher
  int *dl2)                     // The last value

{
  int width_c1;                 // The size of c1 (width)
  int width_c2;                 // The size of c2
  int width_small;              // The size of the smaller character
  int height_c1;                // The height of c1
  int height_c2;                // The height of c2
  float f;

  width_c1 = c1->right - c1->left;
  width_c2 = c2->right - c2->left;
  if (width_c1 < width_c2)
    width_small = width_c1;
  else
    width_small = width_c2;

  if (*dl2 != VERY_SMALL)       // If it's been computed, don't recompute it
  {
    f = ((float) *dl2) / ((float) width_small);
    if (f < FRAC_VALID_DIST)
      return 1;
    else
      return 0;
  }

  height_c1 = c1->bot - c1->top;
  height_c2 = c2->bot - c2->top;

  if (hl2 == 1)         // c1 is higher that c2
  {
    if ((height_c1 * DOT_TO_BODY_HEIGHT_1) < height_c2)
      return comb_dist_ij(c1, c2, hl2, dl2, width_small);
    else
      return comb_dist_noij(c1, c2, hl2, dl2, width_small);
  }
  // Else c2 is higher than c1
  if ((height_c2 * DOT_TO_BODY_HEIGHT_1) < height_c1)
  {
    return comb_dist_ij(c1, c2, hl2, dl2, width_small);
  }
  else
    return comb_dist_noij(c1, c2, hl2, dl2, width_small);


}


////////////////////////////////////////////////////////////////
// This routine sees if larger characters are overlapping enough (left and
// right wise.)  used for identifying type 1 percent signs.
// Returns a 1 if overlapping for a type 1 percent, 0 otherwise
// Assumes that the characters ARE overlapping!
int comb_psign_overlap(
  struct Character *c1,         // The first character
  struct Character *c2)         // The second character
{
  int l1, l2;                   // The length of the characters
  int a1, a2;                   // Computed sides of the character
  struct Character *swap;       // Misc var
  int swapi;                    // Misc var

  l1 = c1->right - c1->left;    // Compute widths
  l2 = c2->right - c2->left;

  if (l1 > l2)                  // c1 is longer
  {
    swap = c1;                  // Swap to make c1 shorter
    c1 = c2;
    c2 = swap;
    swapi = l1;
    l1 = l2;
    l2 = swapi;
  }

  if (((float) l1 / (float) l2 < PERCENT_TYPE1_RMIN)
    return 0;
  if (((float) l1 / (float) l2 > PERCENT_TYPE1_RMAX)
    return 0;

  if (c2->left < c1->left)
    a1 = c1->left;
  else
    a1 = c2->left;
  if (c2->right > c1->right)
    a2 = c1->right;
  else
    a2 = c2->right;

  if ((((float) (a2 - a1)) / ((float) l1) > PERCENT_TYPE1_RATO)
    return 1;
  else
    return 0;
}

////////////////////////////////////////////////////////////////
```

【図155】

```
// This routine will check for the two characters being the last
// part of a percent sign o/    Returns 1 for it's a %, 0 for no.
int comb_left_percent(
  struct Character *c1,        // The characters
  struct Character *c2)
{
// Check for or part of percent :types 2:

  float hl, vl;                // The height and widths of the line
  float hd, wd;                // The height and widths of the dot
  int p;                       // How far from the top to take the slope from

  hl = (float) (c2->bot - c2->top);
  vl = (float) (c2->right - c2->left);
  hd = (float) (c1->bot - c1->top);
  wd = (float) (c1->right - c1->left);

  if (wd * PCNT_PERCENT_OVERLAP <= (float) (c1->right - c2->left))
    // If it overlaps enough

  if (wd * PCNT_DOT_WIDTH_MIN <= wd)
    // If dot is width is not too small

  if (wd * PCNT_DOT_WIDTH_MAX >= wd)
    // If dot is width is not too large

  if (hl * PCNT_DOT_HEIGHT_MIN <= hd)
    // If dot is width is not too small

  if (hl * PCNT_DOT_HEIGHT_MAX >= hd)
    // If dot is width is not too large

  {
    p = (int) (hd * PCNT_TO_TAKE_SLOPE);  // Ignore from top and bottom
    if ((float) (c2->curve2[c2->top+p] - c1->curve1[c1->bot-p]) /
        (float) (c2->bot - c2->top - p - p) >= PCNT_SLOPE)
      return 1;
  }
  return 0;
}

////////////////////////////////////////////////////////////////////////
// This routine will check for the two characters being the 2nd
// part of a percent sign /o    Returns 2 for it's a %, 0 for no.
int comb_right_percent(
  struct Character *c1,        // The characters
  struct Character *c2)
{
  // Check for or part of percent (type 2)

  float hl, vl;                // The height and widths of the line
  float hd, wd;                // The height and widths of the dot
  int p;                       // How far from the top to take the slope from

  hl = (float) (c1->bot - c1->top);
  vl = (float) (c1->right - c1->left);
  hd = (float) (c2->bot - c2->top);
  wd = (float) (c2->right - c2->left);

  if (wd * PCNT_PERCENT_OVERLAP <= (float) (c1->right - c1->left))
    // If it overlaps enough

  if (wd * PCNT_DOT_WIDTH_MIN <= wd)
    // If dot is width is not too small

  if (wd * PCNT_DOT_WIDTH_MAX >= wd)
    // If dot is width is not too large

  if (hl * PCNT_DOT_HEIGHT_MIN <= hd)
    // If dot is width is not too small

  if (hl * PCNT_DOT_HEIGHT_MAX >= hd)
    // If dot is width is not too large

  {
    p = (int) (hd * PCNT_TO_TAKE_SLOPE);  // Ignore from top and bottom
    if ((float) (c1->curve2[c1->top+p] - c2->curve1[c2->bot-p]) /
        (float) (c1->bot - c1->top - p - p) >= PCNT_SLOPE)
      return 1;
  }
  return 0;
}

////////////////////////////////////////////////////////////////////////
// This routine will slide the list over, reading in a new character
// i.e.  1 <- 2 <- 3 <- 4 <- New, (entire list)
void comb_slide(
  struct Combinelist *comb)    // The list
{
  comb->c1 = comb->c2;         // Slide the characters down
  comb->c2 = comb->c3;
  comb->c3 = comb->c4;
  comb->c4 = comb->next;
  if (comb->next != 0)         // Get the next character in the list
    comb->next = comb->next->next;
  comb->hl1 = comb->hl2;       // Slide down heights as well
  comb->hl2 = comb->hl3;
  comb->hl3 = comb->hl4;
  comb->hl4 = -1;              // Set to undefined
  comb->dl1 = comb->dl2;       // Slide down distances
  comb->dl2 = comb->dl3;
  comb->dl3 = comb->dl4;       // Set to undefined
  comb->dl4 = VERY_SMALL;
}

////////////////////////////////////////////////////////////////////////
// This routine will combine two chars (which better be adjacent).  This
// is done by copying c2 and adding it on to the end of c1.
void comb_combine(
  struct Character *c1,        // The first character
  struct Character *c2)        // The second character
{
  int loc;                     // The location of the new character
  int top;                     // The top of the new character
  int ploc;                    // The location divided by 8 (real position)
  int bot;                     // The bottom of the new character
  int count;                   // Used to compute loc
  int s1,s2;                   // Range of overlapping portion of characters
  int left, right;             // The left and right of new character
  int i;

// Combine curve1 and curve2, and compute top and bot
  if ((c1->bot < c2->top) || (c2->bot < c1->top))
  {
    ////////////////// NonOverlapping Chars ////////////////////
    for (i=c2->top; i<c2->bot; i++)   // Copy curves of c2 into c1
    {
      c1->curve1[i] = c2->curve1[i];
      c1->curve2[i] = c2->curve2[i];
    }

    // compute new location
    loc = (c1->loc * (c1->bot - c1->top) + c2->loc * (c2->bot - c2->top)) /
          (c1->bot - c1->top + c2->bot - c2->top);
    ploc = loc/8;

    // Fill blank space between to chars (set to loc, and loc won't change)
    //   (and set top and bot)
    if (c1->bot < c2->top)
    {
      for (i=c1->bot; i<c2->top; i++)
        c1->curve1[i] = c1->curve2[i] = ploc;
      top = c1->top;
```

【図156】

```
    bot = c1->bot;
  } else {
    for (i=c2->bot; i<c1->top; i++)
      c1->curve1[i] = c1->curve1[i] - place;
    top = c1->top;
    bot = c1->bot;
  }
} else {
  ////////////////// Overlapping characters ////////////////////
  // Finishes with position in loc, top and bot
  loc = count = 0;
  // deal with upper non-overlapping range
  if (c2->top > c1->top)      // if c1 is higher
  {
    for (i=c2->top; i < c1->top; i++) // then copy upper part of c2 into c1
    {
      c1->curve1[i] = c2->curve1[i];
      c1->curve2[i] = c2->curve2[i];
      loc += c2->curve1[i] + c2->curve2[i];
    }
    count += c2->top - c1->top;
    top = c2->top;
    c1 = c2->top;
  }
  // deal with lower non-overlapping range
  if (c2->bot < c1->bot)      // If c2 is lower
  {
    for (i=c1->bot; i < c2->bot; i++) // Copy lower part of c1 into c2
    {
      c1->curve1[i] = c2->curve1[i];
      c1->curve2[i] = c2->curve2[i];
      loc += c2->curve1[i] + c2->curve2[i];
    }
    count += c2->bot - c1->bot;
    bot = c1->bot;
    c1 = c2->bot;
  } else {                    // else c1 is lower (or equal to) c2
    for (i=c1->bot; i < c1->bot; i++)
      loc += c1->curve1[i] + c1->curve2[i];
    count += c1->bot - c2->bot;
    bot = c1->bot;
    c1 = c2->bot;
  }
  // Copy in overlapping range
  for (i=c1; i<c2; i++)
  {
    if (c2->curve1[i] < c1->curve1[i])
      c1->curve1[i] = c2->curve1[i];
    if (c2->curve2[i] > c1->curve2[i])
      c1->curve2[i] = c2->curve2[i];
    loc += c1->curve1[i] + c1->curve2[i];
    count++;
  }
  loc = loc * 4 / count;
}

// Compute left and right
if (c1->left < c2->left)
  left = c1->left;
else
  left = c2->left;
if (c1->right > c2->right)
  right = c1->right;
else
  right = c2->right;

// Load information about the stuff.
c1->loc = loc;
c1->top = top;
c1->bot = bot;
c1->left = left;
c1->right = right;
c1->next = c2->next;          // Skip over c2
if (c2->next != 0)
  (c2->next->prev = c1);
c1->type = CHARACTER_COMBINED;
c2->type = CHARACTER_CRASH;
}

////////////////////////////////////////////////////////////////
// This routine will combine characters 1 and 2, and slide them
// out
void comb_combine2(
  struct CombineList *comb)   // The list
{
  comb_combine(comb->c1, comb->c2);
  comb_slide(comb);
  comb_slide(comb);
}

////////////////////////////////////////////////////////////////
//////                                                    //////
//////                The main routine                    //////
//////                                                    //////
////////////////////////////////////////////////////////////////
// This is the main routine to combine a charlist
void charlist_combine(
    CImage *im,               // Image of the line
    struct charlist* clist)   // The character list
{
  struct CombineList comb;    // The 4 characters in the que we're evaluating
  int dummy;

  // Initialize the combination list
  comb.next = clist->first;
  comb_slide(&comb);          // Slide in 4 characters into the list
  comb_slide(&comb);
  comb_slide(&comb);
  comb_slide(&comb);

  // Main loop!  Warning: Indentation is not standard! (Not set too far!
  while(comb.c2 != 0)         // Loop until list is empty
  {
    ////////////////// Combine characters over each other
    // Combine 1 & 2?
    if (comb_might_delta(comb.c1, comb.c2, &(comb.h12)))
    if (comb_dist(comb.c1, comb.c2, comb.h12, &(comb.d12)))
    {
      // Combine 2 & 3? Instead?
      if (!comb.c3 != 0)
      if (comb_might_delta(comb.c2, comb.c3, &(comb.h23)))
      if (comb_dist(comb.c2, comb.c3, comb.h23, &(comb.d23)))
      if (comb.d12 < comb.d23)
      {
```

【図157】

```
// Combine 142 and 141 instead?
if (comb_height_diff(comb.c2, comb.c1, &(comb.h14)))
if (comb_dist(comb.c2, comb.c4, comb.h14, &(comb.d14)))
if (comb.d14 < comb.d13)
{
    // Combine both 142, and 144
    comb_combine12(comb);          // Combine 142 and slide out
    comb_combine14(comb);          // Combine 144 and slide out
    continue;                      // Loop again
}
// Combine chars 143
comb_slide(&comb);
goto_combine12(&comb);
continue;                          // Loop again
}
// Combine chars 142
comb_combine12(&comb);
continue;                          // Loop again
}

//////////////// Now combine parent elms

if ((comb.c1->right > comb.c2->left) && (comb.c2->right > comb.c1->left))
                                   // If overlapping
{                                  // (Don't do if not overlapping)
// Check for parent elm type 1: (almost horizontal line)
if (comb_twin_overlap(comb.c1, comb.c2)) // Combine 1 and 2?
{
  if (comb.c3 != 0)
  if (comb_twin_overlap(comb.c2, comb.c3))
    if (comb_twin_overlap(comb.c1, comb.c3))   // Combine all three characters
    {
      comb_combine(comb.c2, comb.c3);  // Combine 2 and 3
      comb_combine(comb.c1, comb.c2);  // Combine all three characters
      comb_slide(&comb);
      comb_slide(&comb);
      comb_slide(&comb);
      continue;
    }
  comb_combine12(&comb);           // Combine c1 & c2
  continue;
}

// Check for one part of parent (type 2)
if (comb_left_parent(comb.c1, comb.c2))
                                   // It's a or
{
  if (comb.c3 != 0)
  if (comb_right_parent(comb.c2, comb.c3)) // Check for a/o
  {
    comb_combine(comb.c2, comb.c3);      // Combine all three characters
    comb_combine(comb.c1, comb.c2);
    comb_slide(&comb);
    comb_slide(&comb);
    comb_slide(&comb);
    continue;
  }
  comb_combine12(&comb);           // Just combine 1 & 2
  continue;                        // Loop again
}

// Check for /o part of parent
if (comb_right_parent(comb.c1, comb.c2))
{
  comb_combine12(&comb);           // Just combine 1 & 2
  continue;                        // Loop again
}
}

////////////////// Don't combine anything
comb_slide(&comb);                 // Didn't combine anything
}
}
```

```
//
//  Filename: cutdash.C
//  This file contains the routines to remove the dashes from characters
//  in a charlist
//
////////////////////////////////////////////////////////////////////////////
//
//#define PRINT_INFO                    // Print detailed information

////////////////////////////////////////////////////////////////////////////
// The includes
#include"charlist.h"
#include"cimages.h"
#include"mymalloc.h"

#ifdef PRINT_INFO
#include"ms.h"
extern void print_character(cimage4 line, struct Character *chr, int x, int y);
#endif

#define MIN_ABS_WIDTH_TO_CONSIDER 10 // If pixel width less than this, reject.
#define CD_MAX_DASH_HIST 0.3     // x height = max thickness of dash
#define CD_MIN_WIDTH_CONST 0.35 // x height = minimum length of dash
#define CD_MAX_SLOPE_OF_DASH 0.15   // maximum slope of the dash
#define CD_TOO_HIGH 0.3          // x height = max height of dash
#define CD_TOO_LOW 0.8           // x height = max low of dash
                                 //      Note: axis is reversed! (0=up,1=down)

////////////////////////////////////////////////////////////////////////////
// External Routines
void l32_cut_character(cimage4 line, struct Charlist *clist,
                       struct Character **ch, int pj);
void l32_histogram(cimage4 line, struct Character **ch, int *hist);


////////////////////////////////////////////////////////////////////////////
// Check if there is a dash on the left side
int cd_left_dash_cut(
  cimage4 line,                 // The line
  struct Character **ch,        // Pointer to the character we're on (Modified
                                // to skip past newly inserted divisions)
  int *hist)                    // The histogram (previously allocated)
{
  int h;                        // Height of the character
  int t;                        // The thresh crossing point
  int w_min;                    // The minimum width for a dash
  int pl;                       // The break for a left dash
  int wl;                       // The width of the left dash
  int p1, p11, p1h;             // Various points of the dash
  int p2, p21, p2h;             // (See lab log notebook)
  float m;                      // The slope of the dash
  int min_height, max_height;   // The heights of the dashes

  // Compute minimum thickness
  h = (*ch)->bot - (*ch)->top;
  t = (int) (((float) h) * CD_MAX_DASH_HIST);

  // Compute minimum length of dash
  w_min = (int) (((float) h) * CD_MIN_WIDTH_CONST);

  // Compute width of dash
  p1 = (*ch)->left;
  while ((hist[p1] < t) && (p1 < (*ch)->right-1))
    p1++;
  wl = p1 - (*ch)->left;

#ifdef PRINT_INFO
  printf("Height = %d,  threshold = %d,  Min_dash_length = %d\n", h, t, w_min);
  printf("Left Side:\n");
  printf("  P1 = %d  (Wl = %d)\n", p1, wl);
#endif

  // Reject if width is too short
  if ((wl < w_min) || (wl < MIN_ABS_WIDTH_TO_CONSIDER))
  {
    #ifdef PRINT_INFO
    printf("  Rejected: Width of dash is too short.\n");
    #endif
    return 0;
  }

  // Compute p1, p2, p1h, p2h, p11, p21
  p1 = wl/2 + (*ch)->left;                  // Find positions of p1 and p2
  p2 = 3*wl/4 + (*ch)->left;
  for (p1h=(*ch)->top; p1h <(*ch)->bot; p1h++)  // Find p1h
    if ((p1 >= (*ch)->curve1[p1h]) && (p1 < (*ch)->curve2[p1h]))
      if (line.pixel[p1h][p1] & 1)
        break;
  for (p11=(*ch)->bot-1; p11 >= (*ch)->top; p11--)      // Find p11
    if ((p1 >= (*ch)->curve1[p11]) && (p1 < (*ch)->curve2[p11]))
      if (line.pixel[p11][p1] & 1)
        break;
  for (p2h=(*ch)->top; p2h <(*ch)->bot; p2h++)  // Find p2h
    if ((p2 >= (*ch)->curve1[p2h]) && (p2 < (*ch)->curve2[p2h]))
      if (line.pixel[p2h][p2] & 1)
        break;
  for (p21=(*ch)->bot-1; p21 >= (*ch)->top; p21--)      // Find p21
    if ((p2 >= (*ch)->curve1[p21]) && (p2 < (*ch)->curve2[p21]))
      if (line.pixel[p21][p2] & 1)
        break;
  #ifdef PRINT_INFO
  printf("  p1 = %d, p2 = %d\n", p1, p2);
  printf("  p1h, p11  = %d, %d\n", p1h, p11);
  printf("  p2h, p21  = %d, %d\n", p2h, p21);
  #endif

  // Check slope
  m = ((float) (p2h + p21 - p1h - p11))  / ((float) wl);
  if (m < 0)  m = -m;
  #ifdef PRINT_INFO
  printf("  slope = %f\n", m);
  #endif
  if (m > CD_MAX_SLOPE_OF_DASH)
  {
    #ifdef PRINT_INFO
    printf("  Rejected: Dash slopes too much.\n");
    #endif
    return 0;
  }

  // Check height
  max_height = (int) (CD_TOO_HIGH * (float) h) + (*ch)->top;
  min_height = (int) (CD_TOO_LOW * (float) h) + (*ch)->top;

  #ifdef PRINT_INFO
  printf("  Max height = %d, top height = %d\n", max_height, (p1h + p2h)/2);
  #endif
  if ((p1h + p2h)/2 < max_height)
  {
    #ifdef PRINT_INFO
    printf("  Rejected: Dash is too high.\n");
    #endif
```

[図159]

```
    return 0;
  }

  #ifdef PRINT_INFO
  printf("  Min height = %d, bot height = %d\n", min_height, (p11 - p21)/2);
  #endif
  if ((p11 - p21)/2 > min_height)
  {
    #ifdef PRINT_INFO
    printf("  Rejected: Dash is too low.\n");
    #endif
    return 0;
  }

  // It should be a dash!
  return p1-1;
}


//////////////////////////////////////////////////////////////////////
// Check if there is a dash on the right side
int cd_right_dash_cut(
  cimage4 line,                 // The line
  struct Character **ch,        // Pointer to the character we're on (Modified
                                // to skip past newly inserted divisions)
  int *hist)                    // The histogram (previously allocated)
{
  int h;                        // Height of the character
  int t;                        // The thresh crossing point
  int w_min;                    // The minimum width for a dash
  int p1;                       // The break for a left dash
  int w1;                       // The width of the left dash
  int p1, p11, p1h;             // Various points of the dash
  int p2, p21, p2h;             // (See lab log notebook)
  float m;                      // The slope of the dash
  int min_height, max_height;   // The heights of the dashes

  // Compute minimum thickness
  h = (*ch)->bot - (*ch)->top;
  t = (int) (((float) h) * CD_MAX_DASH_HIST);

  // Compute minimum length of dash
  w_min = (int) (((float) h) * CD_MIN_WIDTH_CONST);

  // Compute width of dash
  p1 = (*ch)->right-1;
  while ((hist[p1] < t) && (p1 > (*ch)->left))
    p1--;
  w1 = (*ch)->right - p1;

#ifdef PRINT_INFO
  printf("Right Side:\n");
  printf("  P1 = %d  (W1 = %d)\n", p1, w1);
#endif

  // Reject if width is too short
  if ((w1 < w_min) || (w1 < MIN_ABS_WIDTH_TO_CONSIDER))
  {
    #ifdef PRINT_INFO
    printf("  Rejected: Width of dash is too short.\n");
    #endif
    return 0;
  }

  // Compute p1, p2, p1h, p1h, p11, p21
  p1 = w1/4 + p1;                       // Find positions of p1 and p2
  p2 = 3*w1/4 + p1;
  for (p1h=(*ch)->top; p1h <(*ch)->bot; p1h++)  // Find p1h
    if ((p1 >= (*ch)->curve1[p1h]) && (p1 < (*ch)->curve1[p1h+1]))
      if (line.pixel[p1h][p1] & 1)
        break;
  for (p11=(*ch)->bot-1; p11 >= (*ch)->top; p11--)      // Find p11
    if ((p1 >= (*ch)->curve1[p11]) && (p1 < (*ch)->curve1[p11+1]))
      if (line.pixel[p11][p1] & 1)
        break;
  for (p2h=(*ch)->top; p2h <(*ch)->bot; p2h++)  // Find p2h
    if ((p2 >= (*ch)->curve1[p2h]) && (p2 < (*ch)->curve1[p2h+1]))
      if (line.pixel[p2h][p2] & 1)
        break;
  for (p21=(*ch)->bot-1; p21 >= (*ch)->top; p21--)      // Find p21
    if ((p2 >= (*ch)->curve1[p21]) && (p2 < (*ch)->curve1[p11+1]))
      if (line.pixel[p21][p2] & 1)
        break;
  #ifdef PRINT_INFO
  printf("  p1 = %d, p2 = %d\n", p1, p2);
  printf("  p1h, p11  = %d, %d\n", p1h, p11);
  printf("  p2h, p21  = %d, %d\n", p2h, p21);
  #endif

  // Check slope
  m = ((float) (p2h + p21 - p1h - p11))  / ((float) w1);
  if (m < 0)  m = -m;
  #ifdef PRINT_INFO
  printf("  slope = %f\n", m);
  #endif
  if (m > CD_MAX_SLOPE_OF_DASH)
  {
    #ifdef PRINT_INFO
    printf("  Rejected: Dash slopes too much.\n");
    #endif
    return 0;
  }

  // Check height
  max_height = (int) (CD_TOO_HIGH * (float) h) + (*ch)->top;
  min_height = (int) (CD_TOO_LOW * (float) h) + (*ch)->top;

  #ifdef PRINT_INFO
  printf("  Max height = %d, top height = %d\n", max_height, (p1h + p2h)/2);
  #endif
  if ((p1h + p2h)/2 < max_height)
  {
    #ifdef PRINT_INFO
    printf("  Rejected: Dash is too high.\n");
    #endif
    return 0;
  }

  #ifdef PRINT_INFO
  printf("  Min height = %d, bot height = %d\n", min_height, (p11 + p21)/2);
  #endif
  if ((p11 + p21)/2 > min_height)
  {
    #ifdef PRINT_INFO
    printf("  Rejected: Dash is too low.\n");
    #endif
    return 0;
  }

  // It should be a dash!
  return p1-1;
}
```

【図160】

```
//////////////////////////////////////////////////////////////////
// Check for the dash between 2 characters, and break it if needed
void cd_char(
  cimage& line,                 // The line
  struct Charlist *clist,       // Pointer to charlist
  struct Character **ch,        // Pointer to the character we're on (modified
                                // to skip past newly inserted divisions)
  int *hist)                    // The histogram (previously allocated)
{
  int p;

  // Reject if pixel width is too small.
  if ((*ch)->right - (*ch)->left) <= MIN_LRS_WIDTH_TO_CONSIDER)
    return;

#ifdef PRINT_INFO
  xs_clr_win();
  print_character(line, *ch, 10, 10);
  xs_flush();

  printf("\ncharacter info:\n");
  printf("    top/bot = (%d, %d)\n", (*ch)->top, (*ch)->bot);
  printf("    left/right = (%d %d)\n", (*ch)->left, (*ch)->right);
#endif

  // Compute the histogram
  lll_histogram(line, ch, hist);

  // Break left side?
  p = cd_left_dash_cut(line, ch, hist);
  if (p != 0)
  {
    lll_cut_character(line, clist, ch, p);  // Cut the character
    #ifdef PRINT_INFO
    printf("left dash cut\n");
    printf("\ncharacter info:\n");
    printf("    top/bot = (%d, %d)\n", (*ch)->top, (*ch)->bot);
    printf("    left/right = (%d %d)\n", (*ch)->left, (*ch)->right);
    #endif
  }

  // Break right side?
  p = cd_right_dash_cut(line, ch, hist);
  if (p != 0)
  {
    lll_cut_character(line, clist, ch, p);  // Cut the character
    #ifdef PRINT_INFO
    printf("Right dash cut\n");
    #endif
  }
}


//
//*****************************************************************
// MAIN ROUTINE
//*****************************************************************
//
// This will cut the dashes of of characters
void cut_dash(cimage& line,     // The cimage
    struct Charlist* clist)     // The current charlist
{
  int *histogram;               // The area of the histogram
  struct Character *charon;     // The character we're processing

#ifdef PRINT_INFO
  int dummy;
#endif

  histogram = ivector(line.sizeC());  // Allocate the memory

  charon = clist->first;        //
  while (charon != 0)           // Loop until end of list
  {
    cd_char(line, clist, &charon, histogram);
#ifdef PRINT_INFO
    printf("Enter 0 to continue) ");
    scanf("%d", &dummy);
#endif
    charon = charon->next;      // Get the next character
  }

  free_ivector(histogram);      // Deallocate the memory
}
```

【図195】

```
/*
 * Filename: myalloc.h
 * Header file for myalloc.c
 *
 ---------------------------------------------------------------
 */


/* Function definitions for myalloc.c */
#ifdef CPP
/* Definition for C++ files */
extern "C" {
  int *ivector(int n);
  void free_ivector(int* v);
  float *fvector(int n);
  void free_fvector(float* v);
  int *ivector_ranged(int m, int n);
  void free_ivector_ranged(int* v, int m);
  int **imatrix(int r, int c);
  void free_imatrix(int** m, int r);
  char **cmatrix(int r, int c);
  void free_cmatrix(char** m, int r);
}
#else
/* Definition for C files */
/* The header file for mallocs */
int *ivector();
void free_ivector();
int *fvector();
void free_fvector();
int *ivector_ranged();
void free_ivector_ranged();
int **imatrix();
void free_imatrix();
char **cmatrix();
void free_cmatrix();
#endif
```

【図223】

```
/*-------------------------------------------------------------*/
void zerovpp_terminate(nzlist)
     struct NonZeroVppList *nzlist; /* The list */
{
  free_ivector(nzlist->list);   /* Deallocate memory */
  nzlist->allocated = 0;        /* Set so won't be accidently used */
  nzlist->size = 0;
}

/*-------------------------------------------------------------*/
void zerovpp_add(nzlist, a, b)
     struct NonZeroVppList *nzlist; /* The list */
     int a,b;                   /* Start and stop of the zerovpp */
{
  if (nzlist->size+1 > nzlist->allocated) {
    printf("NonZeroVppList FULL!!!\n");
    exit(1);
  }
  nzlist->list[(nzlist->size)++] = a;
  nzlist->list[(nzlist->size)++] = b;
}
```

```
//
//  Filename: level3a1.C
//  This file contains the routines to do level 3 segmentation (slightly
//  touching) using method 2 (histogram approach)
//
////////////////////////////////////////////////////////////////////////
//


////////////////////////////////////////////////////////////////////
// The Includes
////////////////////////////////////////////////////////////////////
#include"charlist.h"
#include"cimages.h"
#include"myalloc.h"


////////////////////////////////////////////////////////////////////
//  The CONSTANTS
#define MIN_CUT_THRESHOLD 2
#define HISTOBASIC_THRESHOLD 10


////////////////////////////////////////////////////////////////////
// This routine will adjust left, right, curve1, curve2, top, bottom
// and pos to reposition a new character in the charlist, then it
// calls charlist_replace to reposition the new character
// Possible optimization: Usually, this is called when a character is
//   split, so shrinkwrapping one side is 'in vain'
void l3i_shrinkwrap_char(
  cimage& line,                 // The line
  struct Character **ch)        // &Pointer to the character we're on
{
  int top, bot, left, right;    // The new left and right stuff.
  int r,c;
  int loc, loc_count, pos;      // pos = position, loc = pos * 8
  int *curve1, *curve2;         // Shorthand, set to (*ch)->curve1 and 2
  int flag;

  curve1 = (*ch)->curve1;
  curve2 = (*ch)->curve2;

  // Adjust the top of the character, if it needs to move down
  flag = 0;                     // top found when flag == 1;
  for (top = (*ch)->top; (top < (*ch)->bot) && (flag == 0); top++)
    for (c=curve1[top]; c < curve2[top]; c++)
      if (line.pixel[top][c] & 1)   // Stop at the first on pixel
      {
        flag = 1;
        break;
      }

  // Adjust the bottom of the character
  flag = 0;                     // bot found when flag == 1;
  for (bot = (*ch)->bot-1; (bot >= (*ch)->top) && (flag == 0); bot--)
    for (c=curve1[bot]; c < curve2[bot]; c++)
      if (line.pixel[bot][c] & 1) // Stop at the first on pixel
      {
        flag = 1;
        break;
      }
  bot++;                        // Make bot exclusive

  // Now adjust curve1 and curve2, updating left and right to 'shrink'
  // tightly around the border of the character (move the borders up to the
  // edge of the characters.
  loc = loc_count = 0;          // for position of the char (on the fly)
  for (r=top; r<bot; r++)
  {
    while (curve1[r] < curve2[r])       // Loop until they're equal
      if ((line.pixel[r][curve1[r]] & 1)) // If there is a non empty space
                                  // to the right of the left border
        break;                    // Then exit (left border not moved)
      else                        // There's an empty space to the right
        curve1[r] += 1;           // Slide the border over to meet it
    while (curve1[r] < curve2[r])       // Do the same with the right border
      if ((line.pixel[r][curve2[r]-1] & 1))
        break;
      else
        curve2[r] -= 1;

    if (curve1[r] != curve2[r])         // there is a pixel on this line...
    {
      loc += curve1[r] + curve2[r];
      loc_count++;
    }
  }
  if (loc_count != 0)
  {
    loc = loc * 4 / loc_count;  // Compute the position (used for .loc)
    pos = loc/8;                        // Compute the real position
  }

  // Find left and right
  left = curve1[top];
  right = curve2[top];
  for (r=top; r<bot; r++)
  {
    if (curve1[r] < left)
      left = curve1[r];
    if (curve2[r] > right)
      right = curve2[r];
  }

  if (loc == 0)
  {
    loc = 4 * (left+right);
    pos = (left+right)/2;
  }

  for(r=top; r<bot; r++)        // Fill in the blank lines with pos
    if (curve1[r] >= curve2[r])
      curve1[r] = curve2[r] = pos;

  // Copy all the new attributes in
  (*ch)->loc = loc;
  (*ch)->top = top;
  (*ch)->bot = bot;
  (*ch)->left = left;
  (*ch)->right = right;
}


////////////////////////////////////////////////////////////////////
// This routine will cut **ch into two at point p.  clist is updated,
// and **ch returns as the *ch points to the right half of the character
void l3i_cut_character(
    cimage& line,               // The line
    struct Charlist *clist,     // The character list
    struct Character **ch,      // The character to be cut
    int p)                      // Where to cut the character
{
```

```
  struct Character *newchar;    // The new character
  int i;

  newchar = &(clist->chr[clist->size]);  // Set newchar to point to the end
                                         // of the charlist (the next blank character)

  for (i=0; i<clist->vector_height; i++)  // Loop through the char (up-down)
  {
    newchar->curve1[i] = (*ch)->curve2[i];      // Break the character
    newchar->curve1[i] = (*ch)->curve2[i] = p;
    if (newchar->curve1[i] < p)  // Make sure curves of new char are >= p
      newchar->curve1[i] = p;
    if (newchar->curve2[i] < p)
      newchar->curve2[i] = p;
    if ((*ch)->curve1[i] > p)   // Make sure curves of old char are <= p
      (*ch)->curve1[i] = p;
    if ((*ch)->curve2[i] > p)
      (*ch)->curve1[i] = p;

  }
  newchar->top = (*ch)->top;
  newchar->bot = (*ch)->bot;
  newchar->left = p;
  newchar->right = (*ch)->right;
  (*ch)->right = p;

  l32_shrinkwrap_char(line, &newchar);
  Charlist_place_new_char(clist);       // Place newchar into the charlist
  l32_shrinkwrap_char(line, ch);
  Charlist_replace_char(clist, *ch);

  *ch = newchar;                // Adjust ch to the new character
}


////////////////////////////////////////////////////////////////////////
// Compute the histogram of a character
void l32_histogram(             // Compute the histogram
  cimage& line,                 // The line
  struct Character **ch,        // Pointer to the character we're on (Modified
                                // to skip past newly inserted divisions)
  int *hist)                    // The histogram array (output)
{
  int r,c;
  // Zero the histogram
  for (c=(*ch)->left; c < (*ch)->right; c++)
    hist[c] = 0;
  for (r=(*ch)->top; r < (*ch)->bot; r++)       // Compute histogram
    for (c=(*ch)->curve1[r]; c < (*ch)->curve2[r]; c++)
      if (line.pixel(r)[c] & 1)
        hist[c]++;
}


////////////////////////////////////////////////////////////////////////
// Check if a character need to be broken, and break it if so
void l32_insert_breaks(
  cimage& line,                 // The line
  struct Charlist *clist,       // Pointer to charlist
  struct Character **ch,        // Pointer to the character we're on (Modified
                                // to skip past newly inserted divisions)
  int *hist,                    // The histogram (previously allocated)
  int min_cut_threshold,        // The minimum value to make a cut
  int historesis_threshold)     // historesis threshold ( >min_cut_threshold)
{
  int p;                        // The cursor
  int p_end;                    // The end of the cursor
  int min, min_at;              // Where the minimum value is
  int l,r;                      // Left and right boundries of where to search
  int c;

  // Compute the histogram
  l32_histogram(line, ch, hist);

  p_end = (*ch)->right;
  p = (*ch)->left;              // Start at left side
  while (p < p_end)             // Exit when we've scaned the entire histogram
  {
    while (++p < p_end)         // Looking for top-bottom crossing of threshold
      if ((hist[p-1] > historesis_threshold) &&
          (hist[p] <= historesis_threshold))
        break;
    l = p;                      // This is the left side of the area to search

    while (++p < p_end)         // Looking for bottom-top crossing of threshold
      if ((hist[p-1] <= historesis_threshold) &&
          (hist[p] > historesis_threshold))
        break;
    r = p;                      // Remember right edge;

    if (p < p_end)              // If we haven't gone over the edge
    {
      // Search for min in l_to_r range
      min = hist[l];
      min_at = l;
      for (c=l+1; c<r; c++)
        if (hist[c] < min)
        {
          min = hist[c];
          min_at = c;
        }

      // If the min is small enough, cut there
      if (min <= min_cut_threshold)
      {
        l32_cut_character(line, clist, ch, min_at);  // Cut the character
                                // Note: ch is changed to the right char,
                                // which shouldn't mess things up
        printf("Level 1 made a cut.\n");
      }
    }
  }
}


//
//**************************************************************
// MAIN ROUTINE
//**************************************************************
//
// This will do histogram touching on the characters for level 2 method 1

void level1m2(cimage& line,             // The cimage
    struct Charlist* clist)     // The current charlist
{
  int *histogram;               // The area of the histogram
  struct Character *charon;     // The character we're processing

  histogram = ivector(line.sizeC());      // Allocate the memory

  charon = clist->first;        //
  while (charon != 0)           // Loop untill end of list
  {
    l32_insert_breaks(line, clist, &charon, histogram, MIN_CUT_THRESHOLD,
```

[図164]

```
//
// Filename: lineprocess.c
// This routine contains stuff to do character segmentation
//
//////////////////////////////////////////////////////////////
//

#include "charlist.h"
#include "cimages.h"
#include "xs.h"
#include "mymalloc.h"
#include "autoinput.h"

//#define PASS_TO_LEVEL_2       // Defines when runs level 1
//#define PASS_TO_LEVEL_3  // Define if to run level 3
#define PASS_TO_LEVEL_4 // Call the level 4 routines
//#define COMBINING     // Call the combining routines
//#define DASH_CHARS    // Call the routines to cut dashed characters
//#define PAINT_FINAL_CHARLIST

// Declare external routines
void extract_zero_vpp_list(cimage& line, struct NonZeroVppList *list);
void do_outlining(cimage& line, struct Charlist* clist, int start_col,
                  int stop_col);
void do_slightly_touching(cimage& line, struct Charlist* clist, int start_col,
                  int stop_col);
void charlist_combine(cimage& line, struct Charlist*);

// Declare forward referenced internal routines
void print_last_charlist(cimage& line, struct Character *clist, int x, int y);


// Define global variables
int cursor_X, cursor_Y; // Screen coordinates

extern int WINDOW_SIZE_X;
extern int WINDOW_SIZE_Y;

extern int logic_type;          // 0 for none, 1 for OR, 2 for AND-OR

// The external Variables
cimage Process_line;    // The line being segmented (so not reallocated)

// Optimal line routines
extern void cline_OR_3(cimage& oldline, cimage& newline);
extern void cline_OR_3(cimage& oldline, cimage& newline);
extern void cline_AND_12_OR_3(cimage& oldline, cimage& newline);
extern void cline_AND_12_OR_3(cimage& oldline, cimage& newline);

// The compression logic variables (constants)
extern int RESOLUTION_3_LINE_X_AND;
extern int RESOLUTION_3_LINE_C_AND;
extern int RESOLUTION_3_LINE_X_OR;
extern int RESOLUTION_3_LINE_C_OR;
extern int RESOLUTION_3_LINE_X_SCALED;
extern int RESOLUTION_3_LINE_C_SCALED;


//////////////////////////////////////////////////////////////////////
// Misc Utility function
int imin(               // Returns minimum of two values
  int a,
  int b)
{
  if (a>b)
    return b;
  return a;
}


//////////////////////////////////////////////////////////////////////
// Print a character on the screen
void print_character(               // Returns max height used
    cimage& line,                  // The image of the line
    struct Character *chr,         // A pointer to the character
    int x,                         // The position
    int y)
{
  int r,c;
  for (r=chr->top; r < chr->bot; r++)
    for (c=chr->curvel[r]; c < chr->curver[r]; c++)
      if (line.pixel[r][c] & 1)
        xs_plotxy(x + c - chr->left, y + r);

  // Plot the outlined border:
  for (r=chr->top; r<chr->bot; r++)
  {
    xs_plotxy(x + chr->curvel[r] - 2 - chr->left, y + r);
    xs_plotxy(x + chr->curver[r] + 2 - chr->left, y + r);
  }

  // Plot the underline
  for (c=chr->left; c < chr->right; c++)
    xs_plotxy(x + c - chr->left, y + line.sizeR+5);

  // Plot the tag in the underline
  xs_plotxy(x - chr->left + chr->loc/8, y + line.sizeR + 4);
  xs_plotxy(x - chr->left + chr->loc/8, y + line.sizeR - 6);
}


//////////////////////////////////////////////////////////////////////
// Print the last part of a characterlist on the screen (starting at chr)
void print_last_charlist(
    cimage& line,              // The image of the line
    struct Character *chr,
    int x,                     // The position
    int y)
{
  int X_start;                 // The x starting position (for wrap around)

  X_start = x;                 // Remember starting position

  while(chr != 0)              // Loop through entire list
  {

  if (x + chr->right - chr->left > WINDOW_SIZE_X - 20) // Wrap!
    {
     x = X_start;
     y += line.sizeR + 11;
    }
    print_character(line, chr, x, y);
    x += chr->right - chr->left + 10;
    chr = chr->next;           // Get next character
  }
}


//////////////////////////////////////////////////////////////////////
// Print a characterlist on the screen
void print_charlist(
    cimage& line,              // The image of the line
    struct Charlist *clist,    // A pointer to the character list
```

[図165]

```
    int x,                      // The position
    int y)
{
  struct Character *chr;

  chr = clist->first;           // Point to first character
  print_list_charlist(line, chr, x, y);
}


//////////////////////////////////////////////////////////////////////
// Print a nonzerovpplist on the screen
void print_nzvpplist(
    cimage& line,
    struct NonZeroVpplist *nzlist,
    int x,                      // The position
    int y)
{
  int x_start;                  // The x starting position (for wrap around)
  int i,z,c;

  x_start = x;                  // Remember starting position
  for (i=0; i<nzlist->size; i+=2)
  {
    for (z=0; z<line.sizeR; z++)
      for (c=nzlist->list[i]; c<=nzlist->list[i+1]; c++)
        if (line.pixel[z][c] & 1)
          xs_plotxy(x + c, y + z);
    for (z=0; z<line.sizeR; z++)        // Plot double line
    {
      xs_plotxy(x+nzlist->list[i]-1, y+z);
      xs_plotxy(x+nzlist->list[i+1], y+z);
    }

    x += 5;
    if (x > WINDOW_SIZE_X - 50) //  Wrap?
    {
      x = x_start;
      y += line.sizeR + 15;
    }
  }
}


//////////////////////////////////////////////////////////////////////
// This will look at a segment, after segmented from level 1 segmentation
// and decide of it needs to go to level 2!
int decide_level2(      // Returns 1 for go to level 2, 0 otherwise
    cimage& line,       // The image of the line to segment
    int start_col,      // The first column of where the character is (incl)
    int stop_col)       // The last column of this chare\
{
  // Delete following: Removes warnings during compilation
  int a = start_col + stop_col + line.sizeR;

  return 1;
}


//**********************************************************************
//**********************************************************************
//**************          MAIN                    **********************
//**********************************************************************
//**********************************************************************
void process_line(
    cimage& input_line,         // The image of the line to segment
    struct Charlist *clist)     // Output: The characterlist.  Should
                                // be initialized outside this routine
{
  cimage *line_to_process;      // The line to process
  struct NonZeroVpplist *nzlist;
  int i;
  int newr, newc, j,m,n, stop_m, stop_n;
  register char new_pixel_value;

//************************** NO LOGIC ****************************
  line_to_process = &(input_line);              // The default

//************************** THE OR LOGIC ************************
  if (logic_type == 1)
  {
    if (((RESOLUTION_3LINE_R_OR == 2) && (RESOLUTION_3LINE_C_OR == 1)) ||
        ((RESOLUTION_3LINE_R_OR == 3) && (RESOLUTION_3LINE_C_OR == 1)))
    {
      if ((RESOLUTION_3LINE_R_CR == 1) && (RESOLUTION_3LINE_C_OR == 1))
      {
        printf("*** Using optimal OR 2x1 routine\n");
        cline_OR_21(input_line, Process_line);
      }
      if ((RESOLUTION_3LINE_R_CR == 3) && (RESOLUTION_3LINE_C_OR == 1))
      {
        printf("*** Using optimal OR 1x1 routine\n");
        cline_OR_11(input_line, Process_line);
      }
    } else {            // Do standard
      newr = (input_line.sizeR-1)/RESOLUTION_3LINE_R_OR+1;
      newc = (input_line.sizeC-1)/RESOLUTION_3LINE_C_OR+1;
      Process_line.reallocate(newr,newc);

      for (i=0; i<newr; i++)
        for (j=0; j<newc; j++)
        {
          new_pixel_value = 0;
          stop_m = imin((i+1)*RESOLUTION_3LINE_R_OR, input_line.sizeR);
          stop_n = imin((j+1)*RESOLUTION_3LINE_C_OR, input_line.sizeC);
          for (m=i*RESOLUTION_3LINE_R_OR; m < stop_m; m++)
            for (n=j*RESOLUTION_3LINE_C_OR; n< stop_n; n++)
              new_pixel_value = new_pixel_value | input_line.pixel[m][n];
          Process_line.pixel[i][j] = new_pixel_value;
        }
    }
    line_to_process = &(Process_line);
  }

//************************** THE AND OR LOGIC ************************
  if (logic_type == 2)
  {
    if (((RESOLUTION_3LINE_R_AND == 1) && (RESOLUTION_3LINE_C_AND == 2) &&
         (RESOLUTION_3LINE_R_OR == 2) && (RESOLUTION_3LINE_C_OR == 1)) ||
        ((RESOLUTION_3LINE_R_AND == 1) && (RESOLUTION_3LINE_C_AND == 2) &&
         (RESOLUTION_3LINE_R_OR == 3) && (RESOLUTION_3LINE_C_OR == 1)))
    {
      if ((RESOLUTION_3LINE_R_AND == 1) && (RESOLUTION_3LINE_C_AND == 3) &&
          (RESOLUTION_3LINE_R_OR == 2) && (RESOLUTION_3LINE_C_OR == 1))
      {
        printf("*** Using optimal AND 1x2 OR 2x1 routine\n");
        tline_AND_12_OR_21(input_line, Process_line);
      }
      if ((RESOLUTION_3LINE_R_AND == 1) && (RESOLUTION_3LINE_C_AND == 1) &&
          (RESOLUTION_3LINE_R_OR == 3) && (RESOLUTION_3LINE_C_OR == 1))
      {
        printf("*** Using optimal AND 1x2 OR 3x1 routine\n");
        tline_AND_12_OR_31(input_line, Process_line);
      }
```

```
    } else {            // Do standard
      // The AND
      newr = (input_line.sizeR-1)/RESOLUTION_3LINE_R_AND+1;
      newc = (input_line.sizeC-1)/RESOLUTION_3LINE_C_AND+1;
      Process_line.reallocate(newr,newc);

      for (i=0; i<newr; i++)
        for (j=0; j<newc; j++)
        {
          new_pixel_value = 1;
          stop_m = min((i+1)*RESOLUTION_3LINE_R_AND, input_line.sizeR);
          stop_n = min((j+1)*RESOLUTION_3LINE_C_AND, input_line.sizeC);
          for (m=i*RESOLUTION_3LINE_R_AND; m < stop_m; m++)
            for (n=j*RESOLUTION_3LINE_C_AND; n< stop_n; n++)
              new_pixel_value = new_pixel_value & input_line.pixel[m][n];
          Process_line.pixel[i][j] = new_pixel_value;
          }

      // The OR
      newr = (Process_line.sizeR-1)/RESOLUTION_3LINE_R_OR+1;
      newc = (Process_line.sizeC-1)/RESOLUTION_3LINE_C_OR+1;

      for (i=0; i<newr; i--)
        for (j=0; j<newc; j++)
        {
          new_pixel_value = 0;
          stop_m = min((i+1)*RESOLUTION_3LINE_R_OR, Process_line.sizeR);
          stop_n = min((j+1)*RESOLUTION_3LINE_C_OR, Process_line.sizeC);
          for (m=i*RESOLUTION_3LINE_R_OR; m < stop_m; m++)
            for (n=j*RESOLUTION_3LINE_C_OR; n< stop_n; n++)
              new_pixel_value = new_pixel_value | Process_line.pixel[m][n];
          Process_line.pixel[i][j] = new_pixel_value;
        }
      Process_line.reallocate(newr,newc);       // Readjust size afterwards
    }
    line_to_process = &(Process_line);
  }

//********************** SCALED LOGIC **************************
  if (Logic_type == 1)
  {
    newr = (input_line.sizeR-1)/RESOLUTION_3LINE_R_SCALED+1;
    newc = (input_line.sizeC-1)/RESOLUTION_3LINE_C_SCALED+1;
    Process_line.reallocate(newr,newc);
    for (i=0; i<newr; i++)
      for (j=0; j<newc; j++)
        Process_line.pixel[i][j] =
              input_line.pixel[i*RESOLUTION_3LINE_R_SCALED]
                              [j*RESOLUTION_3LINE_C_SCALED];
    line_to_process = &(Process_line);
  }

//********************* END OF LOGIC ROUTINE ***************************

#ifdef VISIBLE
  xs_clr_win();
  (*line_to_process).plot(10,10);
  xs_flush();
#endif


  struct Pointer0ppList masterzlist;   // *** DELETE ***/
  Zero0pp_initialize(&masterzlist, 200);
  nzlist = &masterzlist;        // ***** ADD THIS HIGHER!!! *****/

#ifdef VISIBLE
  xs_clr_win();
  cursor_X = 10;        // The screen coordinates
  cursor_Y = 10;

//  (*line_to_process).plotBox(cursor_X, cursor_Y);
//  xs_flush();
  cursor_Y += (*line_to_process).sizeR + 10;
#endif

  ////////// Insert breaks caused by zero values of VPP
  nzlist->size = 0;             // Zero the list
  extract_zero_vpp_list((*line_to_process), nzlist);

#ifdef PASS_TO_LEVEL_2
  ////////// Insert outlined breaks
  Charlist_zero(clist);                 // Empty the character_list
  for (i=0; i<nzlist->size; i+=2)       // Loop through each character
  {
    if (decide_level2((*line_to_process), nzlist->list[i], nzlist->list[i+1]))
    {
      do_outlining((*line_to_process), clist, nzlist->list[i], nzlist->list[i+1]);
      nzlist->list[i] = -1; nzlist->list[i+1] = -1;
                        // Indicate that this is in a charlist
    }
  }

  ////////// Combine sections that should be combined (in the charlist)
  #ifdef COMBINING
  charlist_combine((*line_to_process), clist);
  #endif
#else
  struct Character *nc;
  // Simply convert the nzlist to the charlist
  Charlist_zero(clist);                 // Empty the character_list
  for (i=0; i<nzlist->size; i+=2)
  {
    nc = &(clist->chr[clist->size]); // Get new_char to where it's going
    nc->top = 0;
    nc->bot = (*line_to_process).sizeR;
    nc->left = nzlist->list[i];
    nc->right = nzlist->list[i+1];
    for (j=0; j<(*line_to_process).sizeR; j++)
    {
      nc->curve1[j] = nzlist->list[i];
      nc->curve2[j] = nzlist->list[i+1];
    }
    Charlist_place_new_char(clist);
  }
#endif


#ifdef PASS_TO_LEVEL_3
  extern void level3a(cimage& line, struct Charlist* clist);

  ////////// Insert slightly touching breaks
  level3a((*line_to_process), clist);
#endif


#ifdef PASS_TO_LEVEL_1
  extern void level1(cimage& line, struct Charlist *clist);

  level1((*line_to_process), clist);
#endif


#ifdef DASH_CHARS
```

【図168】

```
//
// Filename: multiwal.c
// This routine contains stuff to do layer 4 segmentation
//
//////////////////////////////////////////////////////////////////////

extern "C" printbreaks(struct BreakList blist);
extern "C" printbreaks_pri(struct BreakList *blist);
extern "C" int recognized(int **a, int size_r, int size_c);

#include"images.h"
#include"ai.h"
#include"breaklist.H"
#include"hypothesis.h"
#include"charlist.h"
#include"cimages.h"
#include"autoinput.h"

void insert_outline_breaks(pimaget image, BreakList* blist,
                           int bmual, int bmua2);
void print_list_charlist(cimaget image, struct Character *clist, int x, int y);
void fill_whitewrap_charlist(cimaget image, struct Character **ch);

#define MIN_CHAR_WIDTH 4      // The minimum width to be cut
#define MAX_CHAR_WIDTH 51     // The Maximum character width (will
                              // automatically break if width is > this)
                              // (These are approximately in pixels)

#define MAX_CHARS_PER_LINE 100 // Defines maximum number of character in a
                               // piece sent to multilevel

#define MAX_BREAKS 100        // Maximum number of breaks in an image
#define MAX_BREAKS_ACROSS 500 // Maximum number of curvy breaks in image
#define MAX_VPP_SIZE 340      // Maximum image size for VPP array
#define SMOOTH_VPP_FRACTION 0.5 // Fraction of height of VPP to segment

// Define global variables
extern int WINDOW_SIZE_X;
extern int WINDOW_SIZE_Y;

//................ WINDOW CHANGE THESE BACK TO CONSTANTS!!!!! ..........
#define LA_MODE_HEIGHT 0.15   // Minimum valley height to clip left and
                              // Edges after liking rotated projection (as
                              // minimum values on ends will be ignored

int AVP_compute_point(int r, int c, double theta); // Define routine
void print_image_with_breaks(pimaget image, BreakList* blist, int break1,
     int break2, int x, int y);

// Define temporary global variables
int mcursor_r = 30;     // The screen coordinates
int mcursor_c = 30;
int cursor2_r = 10;     // The screen coordinates
int cursor2_c = 10;
int cursor1_r = 10;     // The screen coordinates
int cursor1_c = 10;
int mcursor_col_left = 0;
int mcursor_col_right;
int cursor_top_dline;

//////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////
//                  Find Projection at a rotated angle              //
//////////////////////////////////////////////////////////////////////
// Note: This routine is extremely unoptimized for speed and space!
// Inputs: image, theta,
//         blistwide breaklist, bnum is the break on the left side to start
//                              searching from
//         allocated_A - Highest index allocated for a
// Output: a - VPP, max_r - Highest VPP index value (output).
//         index_out - On what point a[0] corresponds to.
#define P_F_ZERO 10000 // Memory allocated for projection
#define P_F_ZERO 10000
void find_projection(pimaget image, // Where the text is
                     double theta,  // Angle of the text to look (-pi/2..pi/2)
                     struct BreakList* blist, // The breaklist for the image
                     int bnum1,     // break 1 of left side for computing profile
                     int bnum2,     // break 1 of right side for computing profile
                     int C,         // Right point to search for computing profile
                     int allocated_a, // Max sized of all allocated
                     int a[],       // Output: The profile
                     int& max_a,    // Output: The highest index of a used
                     int& index_0a) // Output: What point a[0] corresponds to
{
int store[P_F_STORAGE]; // The profile
int start_store, stop_store; // The highest values used in store
                             // Misc. Vars
int i,j;
int r,c;
int p;

int *leftbreak = new int[image.size1];  // The start and stop locations
int *rightbreak = new int[image.size1];

// Initialize for computing the Angular VPP
for (r=0; r < P_F_STORAGE; r++) store[r] = 0;       // Zero a matrix
start_store = P_F_STORAGE; // Define the bounds to zero
stop_store = 0;
BreakList_return_curves(blist, bnum1, bnum2, leftbreak, rightbreak);

// Compute the Angular VPP
for (r=0; r < image.size1; r++)       // Loop through image
  for (c=leftbreak[r]; c<rightbreak[r]; c++) // Loop through image
    {
      if (c >= C-(image.size1-r)*tan(theta)) // If it's outside the line
        {
          break;
        }
      if (image.pixel[r][c] != 0)
        {                               // Hit from looping through c
          p = AVP_compute_point(r, c, theta);P_F_ERR; // Calculate point
          if ((p < 0) || (p > P_F_STORAGE)) // Check range
            {
              cerr << "Error in find_projection(): P_F_STORAGE too small!\n";
              exit(1);
            }
          store[p]++;                   // Increment count
          if (p < start_store)  start_store = p; // Update max and min range
          if (p > stop_store)   stop_store = p;
        }
    }

// Deallocate memory used
delete [image.size1] leftbreak;
delete [image.size1] rightbreak;

// Copy store to a with information
if (start_store >= P_F_STORAGE) {stop_store = 0;}  // If no projection
{
  max_a = 0; // Return size of 0
```

【図169】

```
    Index_da = 0;  // Set just in case
    return;        // Exit
  }

  MAX_a = step_store - start_store + 1;
  if (max_a > allocated_a)
  {
    cerr << "Error in find_projection. ";
    cerr << "Not enough memory allocated for matrix a[][a]\n";
    exit(1);
  }
  index_da = start_store-P_p_zero;
  for (i=start_steps; i<= step_store; i++)      // Copy store to a
    a[i-start_store] = store[i];
}

// Convert a double to an int by clipping down
// 1.9, -0.3 -> -1, 1.7 -> 1
int clipdouble(double i)
{
  if (i >= 0)
    return (int) i;
  else
    return (int) i-1;
}

// Computes what column of the rotated histogram point (r,c) falls
// in when the axis is rotated theta degrees
int AVPP_compute_point(int r, int c, double theta)
{
  double dr, dc;          // Row and Column
  double di;              // Hist. var (length of diagonal to r,c)
  double dp;              // Output variable
  double alpha;           // Hist. var (angle of (1,0) to (r,c)

  dr = (double) r-.5;     // Convert to double
  dc = (double) c-.5;     //  (the -.5 gets the center of the box)
  alpha = atan(dr/dc);
  di = sqrt(dr*dr + dc*dc);
  dp = di * cos(alpha-theta);
  return clipdouble(dp);
}

//////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////
//////////      The product routine to do thinning          //////////
////////// Overlap: C = A overlapped with B (times) (must be same dimensions)
void product(dimage& A, dimage B, dimage& C)
{
  int i,j;

  C.fill(0, A.size0, A.size1);
  for (i=0; i<A.size0; i++)
    for (j=0; j<A.size1; j++)
      C.pixel[i][j] = A.pixel[i][j] * B.pixel[i][j] / 255.0;
}

//////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////
//////////      This will do thinning                       //////////
//////////                                                  //////////
void thin(dimage& in_image, double sigma, dimage& out_temp)
{
  dimage temp, t1, t2, t3;
  int temptile;
  int i,j;

  ////////// Make the gaussian dimage //////////
  if (sigma < 1.0) sigma = 1.0;          // Can dimage size
  temptile = (int) sigma * 6;
  if ((temptile % 2) == 0)
    temptile++;
  temp.makegauss(sigma, temptile);       // Make dimage

  ////////// Blur degraded image by sigma //////////
  in_image.convolve(in_image, temp);  // in = blurred image

  t2.computeedge(1);                     // t2 = derivative of blurred image
  t2.scale(0,255);

  t3.fill(0,t2.size0, t2.size1); // t3 = inverted derivative of blurred image
  for (i=0; i<t2.size0; i++)
    for (j=0; j<t2.size1; j++)
      t3.pixel[i][j] = 255.0 - t2.pixel[i][j];

  product(out_temp, t3, t3);     // t4 = blurred * derivative (product)
}

//////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////
//////////          Plot one line through a point           //////////
//////////////////////////////////////////////////////////////////////
// (this plots point for point in the y direction (so it is to be
// used with more vertical lines.)
void plot_line_at_angle(
  int r_point,            // The x coord of the point
  int c_point,            // The y coord of the point
  double theta,           // The angle
  int y1,                 // The y range to print
  int y2)
{
  double a, b, xd;
  int x,y;

  // Note: a and b are for x = ay + b !!!!!!!
  a = tan(theta);         // Compute the slope
  b =c_point - a*r_point; // Compute intercept

  for (y=y1; y<y2; y++)
  {
    xd = a * (double) y + b;
    x = (int) xd;
    xm_plotxy(x,y);
  }
}

//////////////////////////////////////////////////////////////////////
//////////          The Segmentation Routines               //////////
//////////////////////////////////////////////////////////////////////
//////////                                                  //////////
//////////          Input breaks where has zero VPP         //////////
//////////////////////////////////////////////////////////////////////
void insert_zerovpp_breaks(
  Breaklist& hits,        // The breaklist to insert breaks in
  int *proj,              // The VPP (previously calculated)
  int proj_size,          // The max value of the VPP
  int proj_zero)          // What value column proj[0] corresponds to
{
```

【図170】

```
int i;                          // The cursor in the projection
int dummy;

for (i=2; i<proj_size; i++)     // For each boarder between points 2:-2 & 3
{
  if ((proj[i-1] != 0) && (proj[i] == 0)) // If right border
    dummy = breaklist_add_break(blist, (i-proj_zero)*blist->vector_height, BREAK_RIGHT_EDGE);
  if ((proj[i-1] == 0) && (proj[i] != 0)) // If left border
    dummy = breaklist_add_break(blist, (i-proj_zero)*blist->vector_height, BREAK_LEFT_EDGE);
}


////////////////////////////////////////////////////////////////////////////
//                                                                        //
//                  Find minimum value in a vector over a specified       //
//                                                                        //
////////////////////////////////////////////////////////////////////////////
void find_minimum_in_vector(
  int a[],                      // The input vector
  int r1,                       // The range to search over (inclusive)
  int r2,
  int &min,                     // Output: minimum value
  int &min_at)                  // Output: It's index
{
  int i;
  min = a[r1];                  // Set first value as default
  min_at = r1;
  for (i=r1+1; i<=r2; i++)
    if (a[i] < min)
    {
      min = a[i];
      min_at = i;
    }
}


////////////////////////////////////////////////////////////////////////////
//                                                                        //
//                  Find minimum in a valley for a rotated profile        //
//                                                                        //
////////////////////////////////////////////////////////////////////////////
void find_min_valley_at_theta(
  glimage& image,               // The image
  struct breaklist* blist,      // The breaklist for the image
  int bound,                    // The range of breaks to search through
  int c,
  double theta,                 // Where to break on the right
  int &min,                     // The angle of rotation to compute
  int &min_at)                  // The minimum value (-1 if no good one)
{                               // Where the minimum is located at (-1 = bad)
  int r1, r2;                   // Range to look for
  int *a, allocated_a;          // The rotated projection
  int angle_index_da;           // Info on the projection
  int max_value_da;             // The maximum value of the projection
  int i;

  // Allocate memory for a[]
  allocated_a = image.size() - 1; // This should be enough
  a = ivector(allocated_a);

  // Find the rotated projection
  find_projection(image, theta, blist, bound, c,
                  allocated_a, a, max_value_da, index_da);

  // Find maximum projection
  max_value_a = a[0];
  for (i=1; i<allocated_a; i++)
    if (a[i] > max_value_a)
      max_value_a = a[i];

  // Choose range to ignore points before first peaks on left and right
  r1 = 0;
  r2 = max_a-1;
  // Find left peak (while slope is still rising) and above IA_IGNORE_WEIGHT
  while ((a[r1] < a[r1+1] || a[r1] < a[r1+2]) || (a[r1] < a[r1+3])) {
    r1++;
    if (r1 >= r2) {             // If reached the left point
      min = min_at = -1;        // Return no minimum found
      free_ivector(a);
      return;
    }
  }
  // Find right peak (while slope is rising (from right to left))
  while ((a[r2] < a[r2-1]) || (a[r2] < a[r2-2]) || (a[r2] < a[r2-3])) {
    r2--;
    if (r1 >= r2) {             // If reached the left point
      min = min_at = -1;        // Return no minimum found
      free_ivector(a);
      return;
    }
  }
  find_minimum_in_vector(a, r1, r2, min, min_at);  // Find the minimum
  min_at += index_da;           // Add the offset from the projection

  free_ivector(a);
}
```

【図171】

```
        allocated_a, max_a, index_0a);

    // Find maximum projection
    max_value_a = a[0];
    for (i=1; i<max_a; i++)
      if (a[i] > max_value_a)
        max_value_a = a[i];

    // Choose range to ignore points before first peaks on left and right
    i1 = 0;
    i2 = max_a-1;

    if (there != 0)
    {
      // Find left peak (while slope is still rising), and above LR_KNOWN_HEIGHT
      while ((a[i2] < a[i2-1]) || (a[i1] < a[i1+1]) ||
             a[i1] < LR_KNOWN_HEIGHT*max_value_a)
        if ((++i1) >= max_a - 2)          // If get to left edge
        {
          min = min_at = -1;
          free_ivector(a);
          return;
        }

      // Find right peak (while slope is rising (from right to left))
      while ((a[i2] < a[i2-1]) || (a[i2] < a[i2-2]) || (a[i2] < a[i2-3]) ||
             a[i2] < LR_KNOWN_HEIGHT*max_value_a)
      {
        i2--;
        if (i2 <= i1 || i2 <=3)           // If reached the left point
        {
          min = min_at = -1;              // Return no minimum found
          free_ivector(a);
          return;
        }
      }

      find_minimum_in_vector(a, i1, i2, &min, &min_at);     // Find the minimum
    }

    // ..... The above is mostly copied from void find_min_valley_at_theta ....
    // ..... The following section is VERY VERY poorly written!!!!!!!!!!!!!!

    for (iscursor_col_left-1; iscursor_col_right-1; i++)  // Draw line above
      xs_plotxy(i, cursor1_y);
    cursor1_y -= 5;

    // Plot the image
    print_image_with_breaks(image, hlist, hnumel, cursor2_x, cursor2_y);

    // Find position of the c line and plot it
    struct Break hb;
                  // Find initial offset
    BreakInit_return_breakinlist(&hb, hnumel, &hl);   // Get first break
    int Cx = cursor2_x - hl.loc/imgm.slsk + c + 2;
    int Cy = cursor2_y - image.slsk + 2;
    plot_line_at_angle(Cy, Cx, theta, cursor2_x - 4, Cy + 5); // Right line

    cursor2_x += image.slsk + 5+1

    // Plot the histogram
    int ttemp = sizeof(histogram)/sizeof(histogram[0]);
    splint(histtemp, hb.hif, theta*(1.0159*180.0));
    xs_drawing(cursor2_x + max_a + 5, cursor2_y + ttemp-4, strings);
    if (theta != 0)                       // Plot dotted line
    {
      int ttt = (int) (LR_KNOWN_HEIGHT * (double) max_value_a);

      for (iscursor2_x - 5; ix cursor2_x + max_a + 4; i++)
        if ((i3 == 0)
          xs_plotxy(i, cursor2_y + ttemp - ttt );
      cursor2_y += ttemp + 2;

      // Plot arrow information at bottom

      j = cursor2_x;
      i1 += cursor2_x; i2 += cursor2_x;                    // NOTE!!!!

      if (there != 0)                     // Plot bar
      {
        for (i=i1; i<=i2; i++)
          xs_plotxy(i, j+1);
        xs_plotxy(i1, j+6); xs_plotxy(i2, j+6);     // Plot ends of bar
        xs_plotxy(i1, j+7); xs_plotxy(i2, j+7);
        xs_plotxy(i1, j+8); xs_plotxy(i2, j+8);
        xs_plotxy(i1, j+9); xs_plotxy(i2, j+9);
      }

      // Draw arrow
      if (theta == 0)
        min_at = bl.loc/image.slsk + 1;
      i = min_at + cursor2_x;
      j = cursor2_y-2;
      xs_plotxy(i, j);
      xs_plotxy(i-1,j+1); xs_plotxy(i+1,j+1); xs_plotxy(i-1,j+1);
      xs_plotxy(i-2,j+2); xs_plotxy(i-1,j+2); xs_plotxy(i,j+2);
                          xs_plotxy(i-2,j+3); xs_plotxy(i-1,j+3);
      xs_plotxy(i, j+4); xs_plotxy(i, j+5);
      cursor2_y += 10;

      free_ivector(a);
    }

//.....
//.....  The find peak and valley routines
//////////////////////////////////////////////////////////////////////////////
// Subroutine to find next peak to the right in a projection
// -1 = none found
int find_peak_to_right(
    int proj[],              // The projection (an array)
    int proj_size,           // The max size of the projection (not inclusive)
    int pos,                 // Where to start looking
    int min_peak_height)     // Minimum value for the peak
{
  int peak;                  // Where the peak is

  peak = -1;                 // Set to not found
  while ((++pos < proj_size) && (peak == -1))
    if ((proj[pos-1] >= proj[pos]) && (proj[pos-1] >= min_peak_height))
      peak = pos - 1;
  return peak;
}

//////////////////////////////////////////////////////////////////////////////
// Subroutine to find next peak to the left in a projection
// -1 = none found
int find_peak_to_left(
    int proj[],              // The projection
    int pos,                 // Where to start looking
    int min_peak_height)     // Minimum value for the peak
{
  int peak;                  // Where the peak is
```

【図172】

```
peak = -1;                          // Set to not found
while ((--pos >= 0) && (peak == -1))
  if ((proj[pos+1] >= proj[pos]) && (proj[pos-1] >= min_peak_height))
    peak = pos + 1;
return peak;
}

////////////////////////////////////////////////////////////////////////
// Subroutine to find next valley to the right in a projection
////////////////////////////////////////////////////////////////////////
int find_valley_to_right(
int proj[],                         // The projection
int proj_size,                      // The max size of the projection (not inclusive)
int pos,                            // Where to start looking
int max_valley_depth)               // Highest point a valley can be
{
int valley;                         // Where the valley is

valley = -1;                        // Set to not found
while ((pos < proj_size) && (valley == -1))
  if ((proj[pos-1] >= proj[pos]) && (proj[pos-1] <= max_valley_depth))
    valley = pos - 1;
return valley;
}

////////////////////////////////////////////////////////////////////////
// Subroutine to find next valley to the left in a projection
// Identical to find_valley_to_right
////////////////////////////////////////////////////////////////////////
int find_valley_to_left(
int proj[],                         // The projection
int pos,                            // Where to start looking
int max_valley_depth)               // Maximum value for the valley
{
int valley;                         // Where the peak is

valley = -1;                        // Set to not found
while ((--pos >= 0) && (valley == -1))
  if ((proj[pos+1] >= proj[pos]) && (proj[pos] <= max_valley_depth))
    valley = pos + 1;
return valley;
}

////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////
//      Compute the Mahrad's angles (given a valley position)         //
////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////
void find_valley_angles(
int* proj,                          // The projection vector
int proj_size,                      // The size of the projection vector
int valley_pos,                     // The position to the right of the 1st peak
double theta1,                      // Angle of left wall
double theta2,                      // Angle of right wall
int min_peak_height,                //
int max_valley_depth)               //
{
int pix, piy;                       // The coord of left peak
int pkx, pky;                       // The coord of left valley bottom
int plx, ply;                       // The coord of right valley
int prx, pry;                       // The coord if right peak

// Find p1: the peak to the left of the valley
pix = find_peak_to_left(proj, valley_pos, min_peak_height);
if (pix == -1)                      // If none found, make it the left side
  pix = 0;
piy = proj[pix];                    // Get its height

// Find pk: the next valley to the left of p1
pkx = find_valley_to_left(proj, pix, max_valley_depth);
if (pkx == -1)                      // If not found, make it the left side
  pkx = pix;
pky = proj[pkx];

// Find p2: the next peak to the right of valley p1
plx = find_peak_to_right(proj, proj_size, pkx, min_peak_height);
if (plx == -1)                      // If not found, make it the right side
  plx = proj_size - 1;
ply = proj[plx];

// Find pr: the nearest valley to the right of peak at p1
prx = find_valley_to_right(proj, plx, max_valley_depth);
if (prx == -1)                      // If not found, make it the left side
  prx = 0;
pry = proj[prx];

theta1 = atan((double) (plx-pix)/ (double) (ply-piy)); // This is positive
theta2 = atan((double) (plx-prx)/ (double) (ply-pry)); // This is negative
}

////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////
//                                                                    //
//                  Mahrad's optimal threshold finding routine        //
//                                                                    //
////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////
void compute_optimal_threshold(
  int* proj,                        // The VPP (an array)
  int proj_size,                    // The size of proj[]
  int level,                        // The threshold level
  int& max_valley_depth,            // The outputs
  int& min_peak_height)
{
int max_value_proj;                 // The maximum value of the projection
int i;
double a, b;
FILE *fp;

if ((fp = fopen("constants.dat", "r")) == NULL)
{
  cout << "\nError in opening constants.data";
  exit(1);
}

fscanf(fp, "%lf", &a);              // Skip ain char width
for (i=0; i<level; i++)
{
  fscanf(fp, "%lf%lf", &a, &b);     // Read a line
  if (a <= -1) {i = level;}         // Exit
  else
  {
    max_valley_depth = -2;          // Return an error
    min_peak_height = -2;
    return;
  }
}
fclose(fp);                         // Close the file

// printf("---- LEVEL %d = %f, %f\n", level, a, b);

// Find maximum projection (for find peak and valley heights)
if (proj_size <= 0)                 // Check for size = 0
{
  max_valley_depth = 0;
  min_peak_height = 0;
  return;
}
```

```
  max_value_proj = proj[0];
  for (i=1; i<proj_size; i++)
    if (proj[i] > max_value_proj)
      max_value_proj = proj[i];

  // Compute min and max valley heights
  max_valley_depth = (int) (a * (double) max_value_proj);
  min_peak_height = (int) (b * (double) max_value_proj);
}


////////////////////////////////////////////////////////////////////////
//////    Reads in the maximum character width from the data file    /////
////////////////////////////////////////////////////////////////////////
int get_max_char_width()
{
  FILE *fp;
  double l;

  if ((fp = fopen("constants.dat", "r")) == NULL)
  {
    cerr << "\nError in opening constants.dat\n";
    exit(1);
  }

  fscanf(fp, "%lf", &l);                  // Read min char width
  fclose(fp);

  return l;
}


////////////////////////////////////////////////////////////////////////
//////      Inserts an angular break into the breaklist           /////
////////////////////////////////////////////////////////////////////////
void insert_angular_break(
  gimage& image,                  // The image
  Breaklist* blist,               // The Breaklist
  int bnum1,                      // The angular range will be forced to
  int bnum2,                      //    fall within these breaks
  double theta,                   // The angle
  int pos)                        // The break position (for line computation!)
{
  int break_position;             // The location of the break in the breaklist
  int *brk;                       // The new break
  int *a1, *a2;                   // Arrays, breaks for bnum1, and bnum2
  int i;

  brk = ivector(image.sizeX);     // Allocate memory for a2
  a1 = ivector(image.sizeX);      // Allocate memory for a1
  a2 = ivector(image.sizeX);      // Allocate memory for a2
  breaklist_return_curves(blist, bnum1, bnum2, a1, a2); // Compute break range

  for (i=0; i<image.sizeX; i++)
  {
    // Compute break position
    brk[i] = (int) ((double) pos / cos(theta) - (double) i * tan(theta));

    // Ensure that it is within range
    if (brk[i] < a1[i])
      brk[i] = a1[i];
    if (brk[i] > a2[i])
      brk[i] = a2[i];
  }

  break_position = (brk[0] + brk[image.sizeX-1]) / 2;

  // Insert the break
  Breaklist_add_break(blist, (break_position)*blist->bvector_height, BREAK_ANGULAR, b
rk);

  free_ivector(brk);
  free_ivector(a1);  // Free memory
  free_ivector(a2);
}


////////////////////////////////////////////////////////////////////////
//  Simple routine used by insert_angular_breaks()
//  Checks a value against the current minimum value.  If less, it updates.
void insert_angular_compare_min(
  int& min,                       // The current minimum
  int& min_at,                    // Where it's at
  double& min_theta,              // It's angle
  int tmin,                       // The test minimum
  int tmin_at,                    //   etc...
  double tmin_theta)
{
  if (tmin_at < 0)                // Don't substitute if tmin_at == -1
    return;
  if (tmin < min)
  {
    min = tmin;
    min_at = tmin_at;
    min_theta = tmin_theta;
  }
}


////////////////////////////////////////////////////////////////////////
//////          Mehrad's angular projection break routine VPP       /////
////////////////////////////////////////////////////////////////////////
// Inserts angular breaks in a segment
int insert_angular_breaks(        // Returns number of breaks inserted
  gimage& image,                  // The image
  Breaklist* blist,               // The breaklist
  int start_break,                // The starting break number
  int stop_break,                 // The stopping break number
  int thresh_level)               // Which set of thresholds to use
{
  int *proj, proj_alloc;          // Projection vector and its size
  int proj_size, proj_zero;       // Highest used value, and what value zero is
  int max_value_proj;             // The maximum value of the projection
  int valley_pos;                 // Position of valley
  int max_valley_depth;           // Maximum height for a valley
  int min_peak_height;            // Minimum height for a peak
  int i, j;                       // Misc var
  double theta1, theta2;          // Angles of valley walls
  int break_count;
  int C;                          // Used for finding angular projections
  int min, min_at;                // A minimum and where it's located,
  double min_theta;               // its angle
  int tmin, tmin_at;              // A temporary min and where it's located

  // Zero number of breaks
  break_count = 0;

  // Allocate the projection vector
  proj_alloc = image.sizeC*3;     // This should be more than enough
  proj = ivector(proj_alloc);
```

[図174]

```
  // Find vertical projection
  find_projection(image, 0, blist, start_break, stop_break, image.sizeC-1,
                  proj_alloc, proj, proj_size, proj_zero);  // Compute VPP

  // Find maximum projection (for find peak and valley heights)
  max_value_proj = proj[0];
  for (i=1; i<proj_size; i++)
    if (proj[i] > max_value_proj)
      max_value_proj = proj[i];

  // Compute max_valley_depth & min_peak_height
  compute_optimal_threshold(proj, proj_size, thresh_level,
        max_valley_depth, min_peak_height);
  if (max_valley_depth < 0)     // Exit if we're too deep in constants
    return break_count;

#ifdef VISIBLE
  // **** PLOT ****
  cursor2_X = cursor1_X;                // Initialize Cursor2 (going down)
  cursor2_Y = cursor1_Y + 30*thresh_level;
  struct Break b1, b2;                  // Compute L and R columns
  BreakList_return_break(blist, start_break, &b1);
  BreakList_return_break(blist, stop_break, &b2);
  cursor1_X -= (b2.loc - b1.loc)/image.sizeX * 40; // Compute width of column
  cursor_col_left = cursor_col_right;   //    (and move cursor 1 to right)
  cursor_col_right = cursor1_X+10;
//  for (i=cursor_top_line; i<WINDOW_SIZE_Y; i++) // Plot double vert. column
//  {
//    xs_plotxy(cursor_col_left-1, i);
//    xs_plotxy(cursor_col_left+1, i);
//  }
  print_image_with_breaks(image, blist, start_break, stop_break,
        cursor2_X, cursor2_Y);          // Print the non-broken image
  cursor2_Y += image.sizeX + 7;
  xs_flush();
  //**** ENDPLOT ****

  // **** PLOT ****:  Plot the histogram
  int tttt = simple_plotmag_hist(proj, proj_size, cursor2_X+3, cursor2_Y);
  int ccccl1 = cursor1_Y + tttt - min_peak_height;      // Plot thresholds
  int ccccl2 = cursor1_Y + tttt - max_valley_depth;
  for (i=0-5; i<proj_size+5; i++)
    if (i % 2)
      xs_plotxy(i+cursor2_X, ccccl1);
    else
      xs_plotxy(i+cursor2_X, ccccl2);
  int ttemp_mcursor_Y = cursor2_Y + tttt - 4;
  cursor1_Y += tttt + 10;
  xs_flush();
  //**** ENDPLOT ****
#endif

  // Find first peak to right
  i = find_peak_to_right(proj, proj_size, 0, min_peak_height);
  if (i == -1)                  // If peak not found...
    return break_count;

  // Find each valley, starting from left position
  while(1)
  {
    // Search for valley to right
    // Note: i should be the peak on the left!
    valley_pos = find_valley_to_right(proj, proj_size, i, max_valley_depth);
    if (valley_pos == -1)               // If not found
      return break_count;

    // Search for next peak to right of valley_pos
    i = find_peak_to_right(proj, proj_size, valley_pos, min_peak_height);
    if (i == -1)                // If not found
      return break_count;

#ifdef VISIBLE
    // **** PLOT **** ***** DELETE HE EVENTUALLY ****** (plot tick for break)
    int tttt1 = find_peak_to_left(proj, valley_pos, min_peak_height);
    for (int j=tttt1; j<=i2; j++)
      xs_plotxy(j+cursor2_X+2, ttemp_mcursor_Y-2);
    xs_plotxy(tttt1+cursor2_X+2, ttemp_mcursor_Y-1);
    xs_plotxy(tttt1+cursor2_X+2, ttemp_mcursor_Y-3);
    xs_plotxy(i2+cursor2_X+2, ttemp_mcursor_Y-1);
    xs_plotxy(i2+cursor2_X+2, ttemp_mcursor_Y-3);
    xs_flush();
    //**** ENDPLOT ****
#endif

    // Now we know a valley exists, so compute angles
    find_valley_angles(proj, proj_size, valley_pos, theta1, theta2,
                       min_peak_height, max_valley_depth);

//    cout << "Position " << valley_pos+proj_zero << ": Angles = " << theta1*180/3.14
159 << ", " << theta2*180/3.14159 << "\n";

    // Place C at the next peak (on the right)
    C = i2+proj_zero;

    // Find minimum of the VPP (theta = 0)
    find_minimum_in_vector(proj, valley_pos, C-proj_zero, &min, &min_at);
    min_theta = 0;
    min_at += proj_zero;

//    cout << "   VPP minimum at " << min_at << " = " << min << "\n";

    // Compute the angular vpp from the last break to C, and find minimum
    find_min_valley_at_theta(image, blist, start_break+break_count,
                  stop_break+break_count, C, theta1, tmin, tmin_at);
//    cout << "   At Theta1, min = " << tmin << "\n";
    // If it has a smaller minimum, remember it
    insert_angular_compare_min(min, min_at, min_theta, tmin, tmin_at, theta1);

    find_min_valley_at_theta(image, blist, start_break+break_count,
                  stop_break+break_count, C, theta2, tmin, tmin_at);
//    cout << "   At Theta2, min = " << tmin << "\n";
    insert_angular_compare_min(min, min_at, min_theta, tmin, tmin_at, theta2);


    ////////////// Check other angles:
    double three_degrees = 3.14159/180.0 * 3.0;
    double six_degrees = 3.14159/180.0 * 6.0;
    double nine_degrees = 3.14159/180.0 * 9.0;

    find_min_valley_at_theta(image, blist, start_break+break_count,
          stop_break+break_count, C, theta1+three_degrees, tmin, tmin_at);
//    cout << "   At Theta1+3, min = " << tmin << "\n";
    insert_angular_compare_min(min, min_at, min_theta, tmin, tmin_at,
                               theta1+three_degrees);
    // Check theta1 - 3
    find_min_valley_at_theta(image, blist, start_break+break_count,
          stop_break+break_count, C, theta1-three_degrees, tmin, tmin_at);
//    cout << "   At Theta1-3, min = " << tmin << "\n";
    insert_angular_compare_min(min, min_at, min_theta, tmin, tmin_at,
                               theta1-three_degrees);
    // Check theta1 + 6
    find_min_valley_at_theta(image, blist, start_break+break_count,
          stop_break+break_count, C, theta1+six_degrees, tmin, tmin_at);
```

【図175】

```
//    cout << '  At Theta1=', min = ' << tmin << '\n';
      insert_angular_compare_min(min, min_at, min_theta, tmin, tmin_at,
                                 theta1-nine_degrees);

      // Check theta1 - 6
      find_min_valley_at_theta(image, blist, start_break+break_count,
          stop_break+break_count, C, theta1-six_degrees, tmin, tmin_at);
      cout << '  At Theta1=', min = ' << tmin << '\n';
//    insert_angular_compare_min(min, min_at, min_theta, tmin, tmin_at,
                                 theta1-six_degrees);

      // Check theta1 + 6
      find_min_valley_at_theta(image, blist, start_break+break_count,
          stop_break+break_count, C, theta1+nine_degrees, tmin, tmin_at);
      cout << '  At Theta1+6, min = ' << tmin << '\n';
//    insert_angular_compare_min(min, min_at, min_theta, tmin, tmin_at,
                                 theta1+nine_degrees);

      // Check theta2 - 3
      find_min_valley_at_theta(image, blist, start_break+break_count,
          stop_break+break_count, C, theta2-three_degrees, tmin, tmin_at);
      cout << '  At Theta2-3, min = ' << tmin << '\n';
//    insert_angular_compare_min(min, min_at, min_theta, tmin, tmin_at,
                                 theta2-three_degrees);

      // Check theta2 + 3
      find_min_valley_at_theta(image, blist, start_break+break_count,
          stop_break+break_count, C, theta2+three_degrees, tmin, tmin_at);
      cout << '  At Theta2+3, min = ' << tmin << '\n';
//    insert_angular_compare_min(min, min_at, min_theta, tmin, tmin_at,
                                 theta2+three_degrees);

      // Check theta1 - 1
      find_min_valley_at_theta(image, blist, start_break+break_count,
          stop_break+break_count, C, theta1-six_degrees, tmin, tmin_at);
      cout << '  At Theta1-6, min = ' << tmin << '\n';
//    insert_angular_compare_min(min, min_at, min_theta, tmin, tmin_at,
                                 theta1-six_degrees);

      // Check theta1 + 6
      find_min_valley_at_theta(image, blist, start_break+break_count,
          stop_break+break_count, C, theta1+six_degrees, tmin, tmin_at);
      cout << '  At Theta1+6, min = ' << tmin << '\n';
//    insert_angular_compare_min(min, min_at, min_theta, tmin, tmin_at,
                                 theta1+six_degrees);

      // Check theta2 - 6
      find_min_valley_at_theta(image, blist, start_break+break_count,
          stop_break+break_count, C, theta2-nine_degrees, tmin, tmin_at);
      cout << '  At Theta2-6, min = ' << tmin << '\n';
//    insert_angular_compare_min(min, min_at, min_theta, tmin, tmin_at,
                                 theta2-nine_degrees);

      // Insert the best break
      cout << '  Final at ' << min_at << ', theta = ' << min_theta*180/3.14159 << ',
      value = ' << min << '\n';

      delete min_at;
#ifdef VISUAL
      find_min_valley_at_theta_print(image, blist, start_break+break_count,
          stop_break+break_count, C, min_theta, tmin, tmin_at);
#endif
```

```
insert_angular_break(image, blist, start_break+break_count,
                     stop_break+break_count, min_theta, min_at);

break_count++;          // Move 1 to the next peak do another peak
}

////////////////////////////////////////////////////////////////////////
// This routine is used as a link to pass a character to recognition
// to see if we need to go to deeper levels of ann.  It returns a
// '0' to go deeper, '1' otherwise  (I know, it's backwards!)
//
int pass_to_deeper_level(
    gImage& image,          // The image
    Breaklist* blist,       // The breaklist
    int b1,
    int b2)
{
    int *a;
    int size_r, size_c;     // The matrix to send to the recognition routine
    int i,j;                // The size of matrix A
    int return_value;       // Used to calculate size_c
    int i,j;
    int *array1, *array2;

    array1 = ivector(image.size(0));
    array2 = ivector(image.size(0));

    breaklist_return_curves(blist, b1, b2, array1, array2);

    r = -1; l = 32000;
    for (int i=0; i<image.size(); i++)
    {
      if (array1[i] < l)  // Get range
        l = array1[i];
      if (array2[i] < l)
        l = array2[i];
      if (array1[i] > r)
        r = array1[i];
      if (array2[i] > r)
        r = array2[i];
    }
    size_r = r - l;
    size_c = image.size();

    if (size_r > MAX_CHAR_WIDTH)  // If too wide, automatic failure
      return 0;

    a = imatrix(size_r, size_c);

    for (i=0; i<size_r; i++)
      for (j=0; j<size_c; j++)
        a[i][j] = 0;

    for (i=0; i<image.size(); i++)
      for (j=array1[i]; j<array2[i]; j++)
        a[i][j-l] = 255;

    return_value = recognize(a, size_r, size_c);

    free_imatrix(a, size_r);
    return return_value;
}
```

```
////////////////////////////////////////////////////////////////////////
////    Recursively insert breaks with higher and higher break levels   ////
////          (Mehrad's "goodcut" routine (I didn't name it))           ////
////////////////////////////////////////////////////////////////////////
int layer4_goodcut(                     // Returns the number of breaks added
  glimages image,                       // The image
  BreakList* blist,                     // The breaklist
  int bnum1,                            // The starting break number
  int bnum2,                            // The stopping break number
  int thresh_level,                     // Which set of thresholds to use
  int perform_recombining)              // == 1 to do recombining, 0 otherwise
{
  int b;                                // Number of breaks added in this routine
  struct Break *brk, *brk2;             // A pointer to the break
  int i,j;
  int R[MAX_CHARS_PER_CHUNK];           // Which piece is recognized, and it's
                                        // corresponding break
  int bn, br;                           // Used to represent start of break in goodcut

  printf("Goodcut Level %d\n", thresh_level);

  // Check to see if width is too long
  brk = BreakList_return_break_ptr(blist, bnum1);
  brk2 = BreakList_return_break_ptr(blist, bnum2);
  if ((brk2->loc - brk->loc) < MIN_CHAR_WIDTH * image.sizeX)
  {
    printf("    Goodcut stoped going up layers because width too small");
    return 0;
  }

  // Insert the breaks
  b = 0;                                // Initialize
  while (b == 0)                        // Loop until break is added
  {
    compute_optimal_threshold(0, 0, thresh_level, i, 1);
    if (i <= -1)                        // If there are no more threshold values to try,
    {
      printf("    Max level reached.  Returning down.\n");
      return 0;                         // Exit and don't go deeper
    }

    // Insert breaks in the image
    b = insert_angular_breaks(image, blist, bnum1, bnum2, thresh_level);
    printf("    %d breaks inserted at level %d.\n", b, thresh_level);
    thresh_level++;
  }

  // Pass each piece which is > MAX_LENGTH to deeper levels
  printf("Passing each piece that is too long to level.\n");
  for (i=bnum1; i<bnum2-b; i++)     // Warning: Deceptive for loop
  {
    brk = BreakList_return_break_ptr(blist, i);
    brk2 = BreakList_return_break_ptr(blist, i+1);
    if ((brk2->loc - brk->loc) > MAX_CHAR_WIDTH * image.sizeX)
    {
      printf("  A break was passed to a higher level because too long\n");
      j = layer4_goodcut(image, blist, i, i+1, thresh_level, 0);
      if (j != 0)                       // If didn't bottom out,
        i += 1;                         // Check each of the new pieces
      b += j;
    }
  }

  if (b == 0)                           // If no breaks inserted
  {
    printf("Couldn't find a place to insert a break.  Exiting goodcut.\n");
    return b;                           // Exit (gone to highest level and can't cut)
  }

  ////////////////////////////
  // GOODCUT METHOD 2 FOLLOWS:  (The recombining)

  if (perform_recombining == 0)
    return b;

  // Pass each piece to recognition  (breaklist range = bnum1 to bnum2-b)

  // Test to see if MAX_CHARS_PER_CHUNK is high enough
  if (bnum2-bnum1+b+1 >= MAX_CHARS_PER_CHUNK)   //
  {
    printf("MAX_CHARS_PER_CHUNK in multilevel.C is too small!\n");
    printf("   In this case it needs to be %d\n", bnum2-bnum1+b+1);
    exit(1);
  }

  // Load in R[], the recognition array: 0 = Not Recognized, 1 = Recognized,
  // and -1 = end of list
  for (i=bnum1; i<bnum2-b; i++)  // Loop through each section
  {
    brk = BreakList_return_break_ptr(blist, i);
    if (brk->next_recog == -1)  // If not previously recognized
      R[i-bnum1] = brk->next_recog = pass_to_deeper_level(image, blist, i, i+1);
  }
  R[bnum2-bnum1+b] = -1;        // End of the Breaklist

  printf("Recognize List:\n  ");
  i = 0;
  while(R[i] != -1)
  {
    if (R[i] == 0)
      printf("N ");
    else
      printf("Y ");
    i++;
  }
  printf("\n");

  printf("Goodcut Logic: \n");
  if (R[0] == 0)                // If first not recognized
  {                             ////////////////////// ON LEFT EDGE ////////////////
    printf("  X K\n");
    if (R[1] == 0)                                  // If NN
    {
      printf("  X N N\n");
      printf("  Trying X (N N) - ");
      if (pass_to_deeper_level(image, blist, bnum1, bnum1+1))   // Try 0-1
      {
        printf("Good\n");
        BreakList_remove_break(blist, bnum1+1);         // 0-1 Good
        b -= 1;
        brk = BreakList_return_break_ptr(blist, bnum1);
        brk->next_recog = 1;
        return b;
      }
      printf("Bad\n");
      if (R[2] == -1)                               // If NNE
      {
        printf("  X N N E - Combining X (N N) E and passing lower\n");
        BreakList_remove_break(blist, bnum1+1);         // Pass 0-1 Deeper
        brk = BreakList_return_break_ptr(blist, bnum1);
        brk->next_recog = -1;
        b -= 1;
        b += layer4_goodcut(image, blist, bnum1, bnum1+1, thresh_level, 1);
```

【図177】

```
return b;
}
if (B[2] == 0)                          // If NSS
{
printf("  E N W NW\n");
printf("  Trying E N W [N M] - \n");
if (pass_to_deeper_level(image, blist, bnum1+1, bnum1+2)) // Try 1-2
{
  printf("Good\n");
  breaklist_remove_break(blist, bnum1+2);      // Combine 1-2
  b -= 1;
  brk = breaklist_return_break_ptr(blist, bnum1+1);
  brk->next_recog = 1;
  return b;
}
printf("Bad\n");
printf("  Trying E [N M N] - ");
if (pass_to_deeper_level(image, blist, bnum1, bnum1+2)) // Try 0-2
{
  printf("Good\n");
  breaklist_remove_break(blist, bnum1+1);
  breaklist_remove_break(blist, bnum1+2);
  b -= 2;
  brk = breaklist_return_break_ptr(blist, bnum1);
  brk->next_recog = 1;
  return b;
}
printf("Bad.\n  Combining E [N M N] E  and passing deeper\n");
breaklist_remove_break(blist, bnum1+1);        // Pass 0-2 deeper
breaklist_remove_break(blist, bnum1+2);
b -= 2;
brk = breaklist_return_break_ptr(blist, bnum1);
b += layer_goodness(image, blist, bnum1, thresh_level, 1);
return b;
}
if (B[1] == 1)                          // If NY
{
printf("  E N Y\n");
if (pass_to_deeper_level(image, blist, bnum1, bnum1+1))  // Try 0-1
{
  printf("Good\n");
  breaklist_remove_break(blist, bnum1+1);      // 0-1 Good
  b -= 1;
  brk = breaklist_return_break_ptr(blist, bnum1);
  brk->next_recog = 1;
  return b;
}
printf("Bad\n");
printf("  Combining E [N Y] and passing deeper\n");  // Pass 0-1 Deeper
breaklist_remove_break(blist, bnum1+1);
b -= 1;
brk = breaklist_return_break_ptr(blist, bnum1);
brk->next_recog = -1;
b -= layer_goodness(image, blist, bnum1, bnum1+1, thresh_level, 1);
return b;
}
if (((B[0] == 0) && (B[2] == 1))                // If NNY
{
printf("  E N Y\n  Trying E [N M N] - ");
if (pass_to_deeper_level(image, blist, bnum1, bnum1+2)) // Try 0-2
{
  printf("Good\n");
  breaklist_remove_break(blist, bnum1+1);
  breaklist_remove_break(blist, bnum1+2);
  brk = breaklist_return_break_ptr(blist, bnum1);
  b -= 2;
  brk = breaklist_return_break_ptr(blist, bnum1);
  brk->next_recog = 1;
  return b;
}
printf("Bad\n  Combining E [N M] and passing deeper\n");
                                        // Pass 0-2 deeper
b -= 2;
b += layer_goodness(image, blist, bnum1+1, bnum1+2, thresh_level, 1);
brk = breaklist_return_break_ptr(blist, bnum1);
brk->next_recog = -1;
return b;
}
printf("  Combining E [N Y] E  and passing deeper\n");
breaklist_remove_break(blist, bnum1+1);
b -= 1;
brk = breaklist_return_break_ptr(blist, bnum1);
brk->next_recog = -1;
b -= layer_goodness(image, blist, bnum1, bnum1+1, thresh_level, 1);
return b;
}
printf("  Trying E [N M] - ");
if (pass_to_deeper_level(image, blist, bnum1, bnum1+1))  // Try 0-1
{
  printf("Good\n");
  breaklist_remove_break(blist, bnum1+1);      // 0-1 Good
  b -= 1;
  brk = breaklist_return_break_ptr(blist, bnum1);
  brk->next_recog = 1;
  return b;
}
printf("Bad\n  Combining E [N M] and passing deeper\n");
                                        // Pass 0-1 deeper
b -= 1;
b += layer_goodness(image, blist, bnum1, thresh_level, 1);
brk = breaklist_return_break_ptr(blist, bnum1);
brk->next_recog = -1;
return b;
}
printf("  Trying E [N Y] - ");
if (pass_to_deeper_level(image, blist, bnum1, bnum1+1))  // Try 0-1
{
  printf("Good\n");
  breaklist_remove_break(blist, bnum1+1);      // 0-1 Good
  b -= 1;
  brk = breaklist_return_break_ptr(blist, bnum1);
  brk->next_recog = 1;
  return b;
}
printf("Bad\n  Combining E [N M] and passing deeper\n");
                                        // Pass 0-1 deeper
b -= 1;
b += layer_goodness(image, blist, bnum1, thresh_level, 1);
brk = breaklist_return_break_ptr(blist, bnum1);
brk->next_recog = -1;
return b;
}
printf("Bad.\n");
printf("  Combining E [N M N] and passing deeper\n");  // Pass 0-2 deeper
breaklist_remove_break(blist, bnum1+1);
breaklist_remove_break(blist, bnum1+2);
b -= 2;
brk = breaklist_return_break_ptr(blist, bnum1);
brk->next_recog = -1;
b -= layer_goodness(image, blist, bnum1, bnum1+2, thresh_level, 1);
return b;
}
if (((B[0] == 0) && (B[1] == 1))                // If NNY
{
printf("  E N Y\n  Trying E [N M N] - ");
if (pass_to_deeper_level(image, blist, bnum1, bnum1+2)) // Try 0-2
{
  printf("Good\n");
  breaklist_remove_break(blist, bnum1+1);
  breaklist_remove_break(blist, bnum1+2);
  brk = breaklist_return_break_ptr(blist, bnum1);
  b -= 2;
  brk = breaklist_return_break_ptr(blist, bnum1);
  brk->next_recog = 1;
  return b;
}
printf("Bad\n  Combining E [N M] and passing deeper\n");
                                        // Pass 0-2 deeper
b -= 2;
b += layer_goodness(image, blist, bnum1+1, bnum1+2, thresh_level, 1);
brk = breaklist_return_break_ptr(blist, bnum1);
brk->next_recog = -1;
return b;
}
printf("  Trying E [N Y] - ");
if (pass_to_deeper_level(image, blist, bnum1, bnum1+1))  // Try 0-1
{
  printf("Good\n");
  breaklist_remove_break(blist, bnum1+1);      // 0-1 Good
  b -= 1;
  brk = breaklist_return_break_ptr(blist, bnum1);
  brk->next_recog = 1;
  return b;
}
printf("Bad\n  Combining E [N M] and passing deeper\n");
                                        // Pass 0-1 deeper
b -= 1;
b += layer_goodness(image, blist, bnum1, thresh_level, 1);
brk = breaklist_return_break_ptr(blist, bnum1);
brk->next_recog = -1;
return b;
}
printf("Bad.\n");
printf("  Combining E [N M Y] and passing deeper\n");  // Pass 0-2 deeper
breaklist_remove_break(blist, bnum1+1);
breaklist_remove_break(blist, bnum1+2);
b -= 2;
b += layer_goodness(image, blist, bnum1, thresh_level, 1);
brk = breaklist_return_break_ptr(blist, bnum1);
brk->next_recog = -1;
return b;
}
if ((B[0] == 1) && (B[1] == 1))                 // If YYY
{
printf("  E N Y\n  Trying E [N M N] - ");
if (pass_to_deeper_level(image, blist, bnum1, bnum1+2)) // Try 0-2
{
  printf("Good\n");
  breaklist_remove_break(blist, bnum1+1);
  breaklist_remove_break(blist, bnum1+2);
  brk = breaklist_return_break_ptr(blist, bnum1);
  b -= 2;
  brk = breaklist_return_break_ptr(blist, bnum1);
  brk->next_recog = 1;
  return b;
}
printf("Bad\n  Combining E [N M] and passing deeper\n");
                                        // Pass 0-2 deeper
b -= 2;
b += layer_goodness(image, blist, bnum1+1, bnum1+2, thresh_level, 1);
brk = breaklist_return_break_ptr(blist, bnum1);
brk->next_recog = -1;
return b;
}
printf("  Trying E [N Y] - ");
if (pass_to_deeper_level(image, blist, bnum1, bnum1+1))  // Try 0-1
{
  printf("Good\n");
  breaklist_remove_break(blist, bnum1+1);      // 0-1 Good
  b -= 1;
  brk = breaklist_return_break_ptr(blist, bnum1);
  brk->next_recog = 1;
  return b;
}
printf("Bad\n  Combining E [N M] and passing deeper\n");
                                        // Pass 0-1 deeper
b -= 1;
b += layer_goodness(image, blist, bnum1, thresh_level, 1);
brk = breaklist_return_break_ptr(blist, bnum1);
brk->next_recog = -1;
return b;
}
printf("Bad.\n");
printf("  Combining E [N M Y] and passing deeper\n");  // Pass 0-2 deeper
breaklist_remove_break(blist, bnum1+1);
breaklist_remove_break(blist, bnum1+2);
b -= 2;
b += layer_goodness(image, blist, bnum1, thresh_level, 1);
brk = breaklist_return_break_ptr(blist, bnum1);
brk->next_recog = -1;
return b;
}
if (B[0] == 1)                          // If YNN
{
printf("  E N W NW\n");
printf("  Trying E [N M] - ");
if (pass_to_deeper_level(image, blist, bn+1, bn+2))
{
  printf("Good\n");
  breaklist_remove_break(blist, bn+2);         // 1-2 Good
  b--;
  brk = breaklist_return_break_ptr(blist, bn+1);
  brk->next_recog = 1;
  return b;
}
printf("Bad\n  Trying [N M N] - ");
if (pass_to_deeper_level(image, blist, bn+1, bn+3))
{
  printf("Good\n");
  breaklist_remove_break(blist, bn+1);         // 0-1 Good
  breaklist_remove_break(blist, bn+3);
  b--;
  return b;
}
printf("Bad\n");
if (B[bn+2] == -1)
```

【図178】

```
{
  printf(" Y N N Yn");                          // YNN
  printf(" Combining [Y N N] E and passing deeper\n");
  BreakList_remove_break(blist, bn+1);          // Pass 0-3 Deeper
  BreakList_remove_break(blist, bn+1);
  BreakList_remove_break(blist, bn+1);
  b--;
  brk = BreakList_return_break_ptr(blist, bn);
  brk->next_recog = -1;
  b -= layer_goodcut(image, blist, bn, bn+1, thresh_level, 1);
  return b;
}
if (!(Rbr+3) == 0)
{
  printf(" Y N N Yn");                          // YNN
  printf(" Trying Y [N N N] - ");
  if (pass_to_deeper_level(image, blist, bn+2, bn+4))
  {
    printf("Good\n");
    BreakList_remove_break(blist, bn+3);        // 2-3 Good
    b--;
    brk = BreakList_return_break_ptr(blist, bn+2);
    brk->next_recog = 1;
    return b;
  }
  printf("Bad\n Trying Y [N N N] - ");
  if (pass_to_deeper_level(image, blist, bn+1, bn+4))
  {
    printf("Good\n");
    BreakList_remove_break(blist, bn+2);        // 1-3 Good
    b-2;
    brk = BreakList_return_break_ptr(blist, bn+1);
    brk->next_recog = 1;
    return b;
  }
  printf("Bad\n Trying [Y N N N] - ");
  if (pass_to_deeper_level(image, blist, bn, bn+4))
  {
    printf("Good\n");
    BreakList_remove_break(blist, bn+2);        // 0-3 Good
    BreakList_remove_break(blist, bn+2);
    BreakList_remove_break(blist, bn+2);
    b-3;
    return b;
  }
  printf("Bad\n Combining Y [N N N] and passing to deeper level\n");
  BreakList_remove_break(blist, bn+2);          // Pass 1-3 Deeper
  BreakList_remove_break(blist, bn+2);
  b--;
  brk = BreakList_return_break_ptr(blist, bn+1);
  brk->next_recog = -1;
  b -= layer_goodcut(image, blist, bn+1, bn+2, thresh_level, 1);
  return b;
}
if (Rbr+2 == 1)                                 // We have YN
{
  printf(" Y N Yn");
  printf(" Trying Y [N Y] - ");
  if (pass_to_deeper_level(image, blist, bn+1, bn+3))
  {
    printf("Good\n");
    BreakList_remove_break(blist, bn+2);        // 1-2 Good
    b--;
    brk = BreakList_return_break_ptr(blist, bn+1);
    brk->next_recog = 1;
    return b;
  }
  printf(" Combining [Y N N] and passing deeper\n");
  BreakList_remove_break(blist, bn+1);          // Pass 0-2 Deeper
  BreakList_remove_break(blist, bn+1);
  b--;
  brk = BreakList_return_break_ptr(blist, bn);
  brk->next_recog = -1;
  b -= layer_goodcut(image, blist, bn, bn+1, thresh_level, 1);
  return b;
}
if (!(Rbr+2) == 0) && (Rbr+2) == 1))            // YNN
{
  printf(" Y N N Yn");
  printf(" Trying Y [N N N] - ");
  if (pass_to_deeper_level(image, blist, bn+1, bn+4))
  {
    printf("Good\n");
    BreakList_remove_break(blist, bn+2);        // 1-3 Good
    b--;
    brk = BreakList_return_break_ptr(blist, bn+1);
    brk->next_recog = 1;
    return b;
  }
  printf(" Combining [Y N N N] and passing deeper\n");
  BreakList_remove_break(blist, bn+1);          // Pass 0-3 Deeper
  BreakList_remove_break(blist, bn+1);
  BreakList_remove_break(blist, bn+1);
  b--;
  brk = BreakList_return_break_ptr(blist, bn);
  brk->next_recog = -1;
  b -= layer_goodcut(image, blist, bn, bn+1, thresh_level, 1);
  return b;
}
printf("WARNING: No pattern matched the goodcut routines.\n");
return b;
}

//////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////
void marker_segmentation(
       gimage& image,          // The input image, a gray image
       BreakList* blist)       // Space allocated for break list
                               //   (This will be initialized)
{
  int first_break, last_break;  // Misc. Used for break numbers
  int *proj; proj_alloc;        // Projection vector and its size
  int b;
  struct Break *brk;

  // Allocate the projection vector
  proj_alloc = image.size();
  proj = ivector(proj_alloc);

  // Find first and last breaks
  first_break = 0;
  last_break = blist->blist_count-1;

  if (last_break != 1)
  {
```

```
    printf("Warning: master_segmentation() (level 0) has somehow been\n");
    printf("         previously segmented!\n");
  }

  // Recognize this piece
  if (pass_to_deeper_level(image, blist, first_break, last_break) != 0)
                        // If recognized
  {
    brk = breaklist_return_break_ptr(blist, 0);
    brk->next_recog = 1;
    free_ivector(proj); // Deallocate memory before exiting
    return;             //  then exit
  }

  // Call the recursive routine, starting at level 0, with recombining
  b = layer4_goodcut(image, blist, first_break, last_break, 0, 1);

  free_ivector(proj);

}


////////////////////////////////////////////////////////////////
void print_image_with_breaks(
  cimage& image,                // The image
  Breaklist* blist,             // The breaklist
  int break1,                   // The range of breaks to print
  int break2,
  int x,                        // The position
  int y)
{
  int r, c, b;
  int start_x = x;              // Starting value of x
  int *start, *stop;            // Start & Stop arrays
  struct Break bi;              // The info on a break
  int last_break;
  int wrap_x;

  wrap_x = WINDOW_SIZE_X - 30;

  if ((break1 < 0) || (break1 >= blist->blist_count) ||
      (break2 < 0) || (break2 >= blist->blist_count))
  {
    cerr << "Error in print_image_with_breaks: Illegal break range:\n";
    cerr << "Selected range is from " << break1 << " to " << break2 << "\n";
    cerr << "Break numbers range from 0 to " << blist->blist_count << "\n";
    int *null = 0;
    (*null) = 1;
    exit(1);
  }

  start = ivector(image.sizeR);
  stop = ivector(image.sizeR);

  // Find initial offset
  Breaklist_return_break(blist, break1, &b1);   // Get first break
  x -= b1.loc/image.sizeR;      // loc = the left side of the printing

  x+=2; y+=2;   // Move over a little for box boundry
  last_break = break1;
  for (b=break1-1; b <= break2; b++) // For each segment
  {
    Breaklist_return_break(blist, b, &b1);      // Get the break
    if (b1.type == BREAK_ERASED)        // Skip erased breaks!
      continue;
    Breaklist_return_curves(blist, last_break, b, start, stop);
    last_break = b;                     // Remember for next time.

    for (r=0; r<image.sizeR; r++)
      for (c=start[r]; c<stop[r]; c++)
      {
        if (image.pixel[r][c] != 0)
          xs_plotxy(x+c,y+r);
        // Draw boarder
        if (r == 0)             // Draw top edge
          xs_plotxy(x+c, y+r-2);
        if (r == image.sizeR-1) // Draw bottom edge
          xs_plotxy(x+c, y+r+2);
        if (c == start[r])      // Draw left edge
          xs_plotxy(x+c-2, y+r);
        if (c == stop[r]-1)     // Draw right edge
          xs_plotxy(x+c+2, y+r);
        if ((r == 0) && (c == start[r]))        // Top left corner
        {
          xs_plotxy(x+c-1, y+r-2);
          xs_plotxy(x+c-2, y+r-2);
          xs_plotxy(x+c-2, y+r-1);
        }
        if ((r == 0) && (c == stop[r]-1))       // Top right corner
        {
          xs_plotxy(x+c+1, y+r-2);
          xs_plotxy(x+c+2, y+r-2);
          xs_plotxy(x+c+2, y+r-1);
        }
        if ((r == image.sizeR-1) && (c == stop[r]-1))   // Bottom right corner
        {
          xs_plotxy(x+c+1, y+r+2);
          xs_plotxy(x+c+2, y+r+2);
          xs_plotxy(x+c+2, y+r+1);
        }
        if ((c == image.sizeR-1) && (c == start[r]))    // Bottom left corner
        {
          xs_plotxy(x+c-1, y+r+1);
          xs_plotxy(x+c-2, y+r+2);
          xs_plotxy(x+c-2, y+r+1);
        }
      }
    if (x+c > wrap_x)
    {
      x -= wrap_x - start_x - 3;
      y += image.sizeR + 10;
    }
    x += 25;    // Slide over to next break is a little to right
  }

  free_ivector(stop);
  free_ivector(start);
}


////////////////////////////////////////////////////////////////
void breaklist_to_charlist(
  cimage& line,                 // The picture of the entire line
  struct Breaklist *blist,              // The breaklist (for this chr)
  struct Charlist *clist,       // The characterlist we're processing
  struct Character **chr,       // The current character we're processing
  int left_side)                // The horiz distance between line and image
{
  struct Character *newchar;    // The new character
  int i,b;

  if (blist->blist_count < 3)   // If no breaks
    return;
```

【図180】

```
for (bx1, b < blist->blist_count-1; b++)
{
    newchar = &(clist->chs[clist->size]);  // Get newchar to point to the end
                                           // of the charlist (the next blank character)

    // Copy curve2 of old char to curve2 of new char
    for (i=(*chr)->top; i<(*chr)->bot; i++)
      newchar->curve2[i] = (*chr)->curve2[i];
    // Copy break into curve2 of old char, and curve2 of new char
    BreakList_return_curve(blist, b, b, &((*chr)->curve2[(*chr)->top]),
                           &(newchar->curve2[(*chr)->top]));

    // Slide over (due to image is a sub-set of line)
    for (i=(*chr)->top; i<(*chr)->bot; i++)
    {
      newchar->curve1[i] += left_side;
      (*chr)->curve2[i] -= left_side;
      if (newchar->curve1[i] > newchar->curve2[i])
        newchar->curve1[i] = newchar->curve2[i];
      if ((*chr)->curve1[i] > (*chr)->curve2[i])
        (*chr)->curve1[i] = (*chr)->curve2[i];
    }

    newchar->top = (*chr)->top;
    newchar->bot = (*chr)->bot;

    CharList_place_new_char(clist,     // Place newchar into the charlist
    lil_strlabcrop_char(line, chr),
    lil_strlabcrop_char(line, &newchar));
    *chr = newchar;
  }
}

//.....
//.....
//.....
//.....
//.....
void main_level(
  Image2 line,          // The entire line
  struct CharList *clist)  // The charlist
{
  Image image;
  struct BreakList blist;
  struct Break *brk;
  Image t1, t2;
  int dummy;
  int r,c,i,j;
  int size_r, size_c;
  struct Character *chr;

  chr = clist->first;   // Point to first character
  while (chr != 0)      // Loop through entire list
  {
#ifdef VISIBLE
    xs_clr_win();
    print_int_charlist(line, chr, 10, 10);
    xs_flush();
#endif
    printf("Enter 0 to pass char to level 4, or number of chars to skip? ");
    i = auto_input_int();
    if (i != 0)
    {
      for (j=0; j<i; j++)
        if (chr != 0)
          chr=chr->next;
    } else {
      printf("Doing level 4 processing\n");
      size_r = chr->bot;
      size_c = chr->right - chr->left;  // Allocate size of matrix
      image.fill(0, size_r, size_c);    // Allocate and zero image

      for (r=chr->top; r < chr->bot; r++) // Copy in the image
        for (c=chr->curve1[r]; c <= chr->curve2[r]; c++)
          if (line.pixel[r][c] < 1)
            image.pixel[r-chr->top][c-chr->left] = 255;

#ifdef VISIBLE
      xs_clr_win();
#endif
      // Now process the image
      mcursor_x = 10;   // The screen coordinates
      mcursor_y = 10;
      cursor1_x = 10;   // The screen coordinates
      cursor2_x = 10;
      cursor1_y = 10;   // The screen coordinates
      cursor_no_last = cursor2_x;
      cursor_y = image.size_r - 30;

      // Insert the left and right breaks based upon curve1 and curve2

      BreakList_initialize(&blist, MAX_BREAKS, MAX_BREADTH, image.size_r);
      dummy = BreakList_add_break(&blist, 0, BACK_PART_OF_IMAGE, 0);
      dummy = BreakList_add_break(&blist, image.size_c*image.size_r, BREAK_END_OF_IMAGE
, 0);
#ifdef VISIBLE
      print_image_with_breaks(image, &blist, 0, blist.blist_count-1, mcursor_x, mcurs
or_y); // Print segmentation
      mcursor_y += image.size_r + 10;
#endif

      makes_segmentation(image, &blist);     // Do the segmentation

      printf("\nFinal Recognition: ");
      for (i=0; i<blist.blist_count-1; i++) {
        brk = BreakList_return_break_ptr(&blist, i);
        if (brk->out_recog == 2)
          printf("? ");
        else
          printf("%d ");
      }
      printf("\n");
#ifdef VISIBLE
      print_image_with_breaks(image, &blist, 0, blist.blist_count-1, mcursor_x, mcurs
or_y); // Print segmentation
      mcursor_y += image.size_r + 10;
      xs_flush();
#endif

      printf("Final goodcut segmentation.  Enter 0 to continue? ");
      j = auto_input_int();

      // Convert the breaklist and incorporate into a charlist
      BreakList_to_charlist(line, &blist, clist, &chr, chr->left);

      BreakList_terminate(&blist);
      chr = chr->next;                   // Get next character
    }
  }
}
```

【図181】

```
//
// Filename: multilevel.c
// This routine contains optimized (12/91) layer 4 segmentation
//
//////////////////////////////////////////////////////////////

//#define PASS_TO_TODAY        // Call Todd's routines (but ignore output)
#define DISABLE_PVP_CACHE       // Disable the PVP Cache

#include"cimages.h"
#include"ra.h"
#include"pvpalloc.h"
#include"charlist.h"
#include"recognpr.h"
#include"math.h"

#define MAX_CHAR_HEIGHT 100  // The maximum allowed line height

//////////////////////////////
// External Variables
//////////////////////////////
extern int VIDEO_STEP_X;
extern int VIDEO_STEP_Y;
extern int MAX_CHAR_LEN;     // The maximum character length in pixels

//////////////////////////////
// Global Variables
//////////////////////////////

#ifdef PASS_TO_TODAY
extern "C" todd_recognize(int r, int c, char** image);
#endif

// The rotation table variables
#define ROT_MAX_ANGLE 15     // The maximum degree range (inclusive)
#define ROT_ANGLE_RESOLUTION 3  // The distance in degrees
#define ROT_NUMBER_ANGLES 11    // = (ROT_MAX_ANGLE*2/ROT_ANGLE_RESOLUTION)
int **Rot;                   // The entire projection array
                             //   Rot[0][0] = Projection array when
                             //   height 0 and degree = (0-1)*(ROT_MAX_ANGLE/ROT_ANGLE_RESOLUTION)

#define LR_EDGE_HEIGHT 0.25  // Minimum valley height to clip left and
                             // Edges after taking rotated projection (as
                             // minimum values on ends will be ignored

// The projection array (allocated once per line; a lot of routines use it)
int *Proj;                   // Allocated memory for projection
int *PP;                     // Allocated memory for the Vertical projection

// The constants loaded from constants.dat
int MAX_CHAR_LEN;            // The size of Count1 & Count2
float *Count1;               // The maximum valley depth ratio
float *Count2;               // The minimum peak height ratio

// The PVP Cache variables
#define PVP_CACHE_SIZE 0     // The size
int PVP_Level;               // -1 if the cache is empty (set this to clear)
int PVP_I;                   // The last element in the cache
int PVP_J;                   // The row past the last element
int PVP_R1[PVP_CACHE_SIZE];  // Array of r1's
int PVP_R2[PVP_CACHE_SIZE];  // Array of r2's
int PVP_V1[PVP_CACHE_SIZE];  // Array of v1's
int PVP_H1[PVP_CACHE_SIZE];  // Array of h1's
int PVP_Loc[PVP_CACHE_SIZE]; // Location of cuts
int PVP_Angle[PVP_CACHE_SIZE]; // Angle of the cut

// Global Curses (for plotting) variables
int CursesLin;               // Line display
int CursesCol;               // The final output display
int CursesTop;               // The top of the column going across
int CursesBot;               // The column going down
int Curses_PP_Hack; // Where to plot lines under histogram

//..............  Temp Test Routines  ..................
// Print Charlist in useful format
extern "C" void Print_Useful_Charlist(
  struct Charlist *clist,
  struct Character *ch)
{
  struct Character *chon;
  int i;

  chon = clist->first;
  while(chon)
  {
    if (chon == ch) printf("*");
    printf("%d-%d", chon->left, chon->right);
    if (chon == ch) printf("*");
//    printf("\n  curval:");
//    for (i=chon->top; i<chon->bot; i++)
//      printf("%d ",chon->curval[i]);
//    printf("\n  curve:");
//    for (i=chon->top; i<chon->bot; i++)
//      printf("%d ",chon->curve2[i]);
//    if (chon->seg == FUSION_RESOLUTION)
//      printf("\n");
    if (chon->seg != 1) FAILED_RECOGNITION;
    printf(" Allweight=%d",chon->level_dif_at);
    if (chon->xd == UNRECOGNIZED)
      printf(" U");
    printf("\n");

    if (chon->next != 0)
    {
      if (chon->next->prev != chon)
      {
        printf("\n*** Charlist link error: ch->next->prev != ch !\n");
        exit(1);
      }
      chon = chon->next;
    }
    else
    {
      printf("\n");
      return;
    }
  }
}

//..................  Simple Utility Functions  ..................
// Return absolute value of an int
int abs(int i)
{
  return (i < 0) ? -i : i;
}

//////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////
// Initialize Count1 and Count2 (read from constants.dat)
void Initialize_Constants()
{
  int i;
  float *a;                  // Temporary storage
  float *b;                  // Temporary storage
  FILE *fp;
```

【図182】

```
if ((fp = fopen("constants.dat", "r")) == NULL)
{
  cerr << "\nError in opening constants.data";
  exit(1);
}

// *** This should change before the final version, when we know the size
//     of constants.dat, hardcode it!
max_thresh_level = 0;
a = (vector(200));
b = (vector(200));

for (i=0; i<200; i++)
{
  fscanf(fp, "%lf", &(a[i]), &(b[i]));
  if ((a[i] < 0) || (b[i] < 0))
  {
    max_thresh_level = i;
    break;
  }
}

if (i>=200)
{
  printf("*** Error: Only 200 entries in constants.dat is allowed!  Change the code
in");
  exit(1);
}

Const1 = (vector(max_thresh_level));
Const2 = (vector(max_thresh_level));
for (i=0; i<max_thresh_level; i++)
{
  Const1[i] = a[i];
  Const2[i] = b[i];
}

free_(vector(a));
free_(vector(b));
}

////////////////////////////////////////////////////////////////////////
// Initialize rotation.dat
// :This must be called before calling compute_rotated_projection
//
void ik_initialize_rot()
{
  int d_index;          // Index of degree we're working on
  double d_r_d;         // The number of degrees, and Laguerre term
  int i;                // Looping var; for dr. and degree on
  int a,b;

  a = ROT_INDEX_RANGE;
  b = MAX_CHAR_HEIGHT;
  Rot = (matrix(a, b));

  for (d = -ROT_INDEX_RESOLUTION; d<=ROT_MAX_RANGE; d+=ROT_INDEX_RESOLUTION)
  {
    d_index = ((int) (d/ROT_INDEX_RESOLUTION+.5)) - 1;
    d_d = tan(d * 3.14159/180.0);
    for (i=0; i< MAX_CHAR_HEIGHT; i++)
      Rot[d_index][i] = (int) (d_d*((double) i+.5);
  }
}

//.....
void ik_plot_char(
  image* image,         // The image
  struct Character *ch, // The character to plot
  int x,                // The upper left hand corner (.left, .top) to plo
  int y)
{
  int z,c;
  for (z=ch->top; z < ch->bot; z++)
  {
    for (c = ch->curve1[z]; c< ch->curve2[z]; c++)
      if (image.pixel[z][c] & 1)
//      xm_plotxy(xwin->curve1[z]-ch->left, yz);
//      xm_plotxy(xwin->curve2[z]-ch->left, yz);
      ;
    xm_flush();
  }
}

void ik_plot_charlist(
  image* line,          // The image
  struct Charlist *clist,  // The clist
  int x,                // The position
  int y)
{
  struct Character *ch;
  int x_start;          // The x stating position

  x_start = x;
  ch = clist->first;
  while(ch != 0)
  {
    if (x + ch->right - ch->left > WINDOW_SIZE_X - 20) // Wrap?
    {
      x = x_start;
      y += line.size4 + 15;
    }
    ik_plot_char(line, ch, x, y);
    x += ch->right - ch->left + 10;
    ch = ch->next;      // Get next character
  }
}

// This routine will plot a character in the lower left hand corner
// (for the purposes of displaying to prompt for recognition)
void ik_plot_lower_left(
  image* image,         // The image
  struct character *ch)
{
  int z,c;
  int delta;
  delta = ch->bot - ch->top;

  for (c=0; c<(ch->right-ch->left); c++)        // Erase area
    for (z=WINDOW_SIZE_Y-delta-15; z < WINDOW_SIZE_Y; z++)
      xm_plotxy(c,z);

  for (z=ch->top; z < ch->bot; z++)
    for (c=ch->curve1[z]; c < ch->curve2[z]; c++)       // Plot char
      if (image.pixel[z][c] & 1)
        xm_plotxy(c-ch->left, WINDOW_SIZE_Y-2-delta+z-ch->top);

  for (c=0; c<(ch->right-ch->left); c++)        // Plot box around it
```

【図183】

```
{
    xs_plotxy(c,WINDOW_SIZE_Y-1);
    xs_plotxy(c,WINDOW_SIZE_Y-delta-1);
}
for (r=WINDOW_SIZE_Y-delta-4; r < WINDOW_SIZE_Y; r++)
{
    xs_plotxy(0, r);
    xs_plotxy(3+ch->right-ch->left, r);
}
xs_flush();
}

// This routine will plot a range of characters in the lower left hand corner
// (for the purposes of displaying to prompt for recognition)
// Note, they MUST have the same height range!!!
void ld_plot_lower_left_2chars(
    charset image,          // The image
    struct Character *ch1,  // Left character
    struct Character *ch2)  // Right character
{
int r,c;
int delta;

if ((ch1->top != ch2->top) || (ch1->bot != ch2->bot))
{
    printf("***** ERROR: ld_plot_lower_left_2chars() called with invalid
    printf("top's and bottoms!!!! ch1(t,b) = (%d,%d)\n",
                ch1->top, ch1->bot, ch2->bot);
    exit(1);
}
delta = ch1->bot - ch1->top;
for (c=0; c<ch1->right-ch1->left; c++)          // Erase area
    for (r=WINDOW_SIZE_Y-delta-1; r < WINDOW_SIZE_Y; r++)
        xs_unplotxy(c,r);
for (r=ch1->top; r < ch1->bot; r++)
    for (c=ch1->curve1[r]; c < ch2->curve2[r]; c++)     // Plot char
        if (image.pixel[r][c] & 1)
            xs_plotxy(2+c-ch1->left, WINDOW_SIZE_Y-3-delta+r-ch1->top);
for (c=0; c<4+ch2->right-ch1->left; c++)
                                    // Plot box around it
{
    xs_plotxy(c,WINDOW_SIZE_Y-1);
    xs_plotxy(c,WINDOW_SIZE_Y-delta-4);
}
for (r=WINDOW_SIZE_Y-delta-4; r < WINDOW_SIZE_Y; r++)
{
    xs_plotxy(0, r);
    xs_plotxy(3+ch2->right-ch1->left, r);
}
xs_flush();
}

// This routine will start a new column on the display
void ld_plot_new_column(
    int n)          // n == 0 for double line, == 1 for single line
{
    int r;
    Cursor1_X += 10;
    Cursor2_X = Cursor1_X;
    Cursor1_Y = Cursor2_Y;
    if (Cursor1_X > (WINDOW_SIZE_X - 10))
    {
        printf("Going to clear window; Enter 0 to continue! ");
        scanf("%d", &r);
        xs_cl_vis();
        Cursor1_X = Cursor2_X = 20;
    }
    if (n)
    {
        for (r=Cursor1_Y; r<WINDOW_SIZE_Y; r++)
            xs_plotxy(Cursor2_X-10,r);
    }
    else
    {
        for (r=Cursor1_Y; r<WINDOW_SIZE_Y; r++)
        {
            xs_plotxy(Cursor2_X-11,r);
            xs_plotxy(Cursor2_X-9,r);
        }
    }
    Cursor2_X += 40;            // Minimum of 40 columns over
    xs_flush();
}

//.....  TOP RECOGNITION ROUTINE .....
// Recognized character defined by from ch1 to ch2 (inclusive)
int recognition(                // Returns 0 for unrecognized, 1 otherwise
    charset image,              // The image
    struct Character *ch1,      // Left character
    struct Character *ch2)      // Right character
{
#ifdef VISIBLE
    ld_plot_lower_left_2chars(image, ch1, ch2);
#endif

#ifdef PASS_TO_TOSHI
    int size_r, size_c;
    int r,c;
    char **p;
    size_r = ch1->bot - ch1->top;
    size_c = ch2->right - ch1->left;
    p = matrix(size_r, size_c);
    for (r=0; r<size_r; r++)
        for (c=0; c<size_c; c++)
            p[r][c] = 0;
    for (r=ch1->top; r<ch1->bot; r++)
        for (c=ch1->curve1[r]; c<ch2->curve2[r]; c++)
            p[r-ch1->top][c-ch1->left] = image.pixel[r][c];
    toshi_recognize(&size_r, size_c, p);
    free_matrix(p, size_r);
#endif

    if ((ch2->right - ch1->left) > MAX_CHAR_LEN)
    {
        printf("Character unrecognized because it was too long\n");
        return 0;
    }
    printf("Recognized (1-yes, 0=no) ");
    return data_input_int();
}

//////////////////////////////////////////////////////////////////////
// This routine will determine if a character is too long (and thus exits)
int ld_too_long(                // Return 1 if too long, 0 otherwise
    struct Character *ch1,      // Left character
    struct Character *ch2)      // Right character
{
    if ((ch2->right - ch1->left) > MAX_CHAR_LEN)
    {
        return 1;
    }
    return 0;
}

//.....  Character Utility Routines .....
// This routine will move ch->left, and ch->right so they are
//  set to the 1st and last "on" pixel in the image.  This is used to
```

```
//   eliminate white space from the left and right side of a character
//   after it is cut so the size won't be too inaccurate.
//   Call when place the cut.  ch1 = left char, ch2 = right char
void ld_char_reset_lr(
    cimage& image,
    struct Character *ch1,      // The left character (only does right side)
    struct Character *ch2)      // The right character (only does left side)
{
  int r,c;
  int new_left, new_right;      // The new edges of the characters

  // Right side of char1:  Look for 1st on pixel
  for (c = ch1->right-1; c>=ch1->left; c--)
    for (r = ch1->top; r<ch1->bot; r++)
      if ((c >= ch1->curve1[r]) && (c < ch1->curve2[r]))
        if (image.pixel(r)[c] & 1)      // If pixel is on
        {
          new_right = c+1;
          goto exit_loop;                       // Break out of the loop
        }
  exit_loop:

  // Set right side of character to this value
  ch1->right = new_right;
  for (r=ch1->top; r<ch1->bot; r++)
    if (ch1->curve2[r] > new_right)
    {
      ch1->curve2[r] = new_right;
      if (ch1->curve1[r] > new_right)
        ch1->curve1[r] = new_right;
    }

  // Do the left side of ch2
  // Look for the 1st on pixel
  for (c=ch2->left; (c<ch1->right); c++)
    for (r=ch2->top; r < ch2->bot; r++)
      if ((c >= ch2->curve1[r]) && (c < ch2->curve2[r]))
        if ((image.pixel[r][c] & 1) == 1)
        {
          new_left = c;
          goto exit_loop2;              // Break out of the loop
        }
  exit_loop2:

  // Set the left side of character to this value
  ch2->left = new_left;
  for (r=ch2->top; r<ch2->bot; r++)
    if (ch2->curve1[r] < new_left)
    {
      ch1->curve1[r] = new_left;
      if (ch1->curve2[r] < new_left)
        ch2->curve2[r] = new_left;
    }
}


//*************** THE ROTATED PROJECTION ROUTINES ***************
void ld_find_projection(
    cimage& image,      // The input image
    struct Character *ch, // The character to find the projection on
    int angle,          // The rotation/ROT_DEG_RESOLUTION. Must be within a
                        //   valid range!
    int range1,         // The column range to search through
    int range2,         //
    int *ang_proj,      // The matrix to put the projection into (*MUST*
                        //   be previously allocated)
    int *ang_proj_min,  // The min index of ang_proj[] (inclusive) (output)
    int *ang_proj_max)  // The max index of ang_proj[] (exclusive) (output)
                        // Note: Projection put in Proj[]
{
  int r,c;              // Looping variables

  // Do a standard histogram
  if (angle == 0)       // Do a standard vpp histogram
  {
    *ang_proj_min = range1;
    if (*ang_proj_min < ch->left)
      *ang_proj_min = ch->left;
    *ang_proj_max = range2;
    if (*ang_proj_max > ch->right)
      *ang_proj_max = ch->right;

    for (c=ch->left; c<=ch->right; c++)  // Zero projection array
      ang_proj[c] = 0;
    for (r=ch->top; r<ch->bot; r++)      // Compute the projection
      for (c=ch->curve1[r]; c<ch->curve2[r]; c++)
        if (image.pixel[r][c] & 1)
          (ang_proj[c])++;
    return;
  }

  // Do a theta is greater than zero
  if (angle > 0) {
    angle--;                    // Get the index to use in Rot
    *ang_proj_min = range1;
    if (*ang_proj_min < ch->left)
      *ang_proj_min = ch->left;
    *ang_proj_max = range2;
    if (*ang_proj_max > ch->right)
      *ang_proj_max = ch->right;
    *ang_proj_min -= Rot[angle][ch->bot-ch->top];

    for (c=*ang_proj_min; c<*ang_proj_max; c++) // Zero projection array
      ang_proj[c] = 0;
          for (r=ch->top; r<ch->bot; r++)       // Compute the projection
      for (c=ch->curve1[r]; c<ch->curve2[r]; c++)
        if (image.pixel[r][c] & 1)
          (ang_proj[c-Rot[angle][ch->bot-r-1]])++;
    return;
  }

  // Do a theta is less than zero
  angle = -angle - 1;           //-Get the index to use in Rot
  *ang_proj_min = range1;
  if (*ang_proj_min < ch->left)
    *ang_proj_min = ch->left;
  *ang_proj_max = range2;
  if (*ang_proj_max > ch->right)
    *ang_proj_max = ch->right;
  *ang_proj_max += Rot[angle][ch->bot-ch->top];

  for (c=*ang_proj_min; c<*ang_proj_max; c++)   // Zero projection array
    ang_proj[c] = 0;
  for (r=ch->top; r<ch->bot; r++)       // Compute the projection
    for (c=ch->curve1[r]; c<ch->curve2[r]; c++)
      if (image.pixel[r][c] & 1)
        (ang_proj[c+Rot[angle][ch->bot-r-1]])--;

  return;
}
```

【図185】

```
//.....
// This routine will insert an entry into the PVP cache (a cyclic cache)
// with inputs p1,v1,p2 and output loc,angle.
void CPVP_CACHE_insert(  // The top of 1st peak
    int p1,              // The start of valley
    int v1,              // The end of valley
    int p2,              // The top of 2nd peak
    struct *ch,          // The character we're in
    int *min_theta,      // The final angle
    int min_at)          // Where the minimum is at
{
#ifdef DISABLE_PVP_CACHE
    return;
#endif

    int f;

    // Make sure the PVP area doesn't have a cw running through it!
    for (f=p1+1; f < ch->hbc; f++)
      if ((ch->curve1[f] > p1) || (ch->curve2[f] < p2))
                         // If so, don't put on PVP Que
        return;

    if (PVP_Zero)
    {
      PVP_f = 0;         // Insert 1st element
      PVP_f = 1;
      PVP_Zero = 0;
    } else {
      if (PVP_f == PVP_B     // List full, wrap last out
      {
        // The next line is basically PVP_f = (PVP_f-1) % CACHE_SIZE
        PVP_f = (PVP_f==0) ? PVP_CACHE_SIZE-1 : PVP_f-1;
        PVP_B = PVP_f;
      } else {
        PVP_f = (PVP_f==0) ? PVP_CACHE_SIZE-1 : PVP_f-1;
      }
    }
    // Now place the new data at PVP_f
    PVP_p1[PVP_f] = p1;
    PVP_v1[PVP_f] = v1;
    PVP_v2[PVP_f] = v2;
    PVP_p2[PVP_f] = p2;
    PVP_loc[PVP_f] = min_at;
    PVP_angle[PVP_f] = min_theta;
}

// This routine will check PVP against past results and if we've done
// it before, it will just grab the results.
// Return 0 if not hit, 1 if hit
int CPVP_Cache_Hit(  // The top of 1st peak
    int p1,          // The start of valley
    int v1,          // The end of valley
    int p2,          // The top of 2nd peak
    struct *ch,      // The character we're in
    int *min_theta,  // Output: The final angle
    int *min_at)     // Output: Where the minimum is at
{
    int i,r;

#ifdef DISABLE_PVP_CACHE
    return 0;
#endif

    if (PVP_Zero)      // Return 0 if also
      return 0;

    i = PVP_f;
    do
    {
      if ((PVP_p1[i] == p1) && (PVP_v1[i] == v1) &&    // If hit!
          (PVP_v2[i] == v2) && (PVP_p2[i] == p2))
      {
        // Ensure the PVP area doesn't have a cw running through it
        for (r=ch->hbc; r < ch->hbc; r++)
          if ((ch->curve1[r] > p1) || (ch->curve2[r] < p2))
            return 0;          // If so, return a miss
        *min_theta = PVP_angle[i];
        *min_at = PVP_loc[i];
        return 1;
      }
      i--;
      if (i == -1) i = PVP_C;
    } while (i != PVP_f);      // Loop until it hits the end
    return 0;                  // None found
}

//.....
//..... The find peak and valley routines
//..... Subroutine to find next peak to the right in a projection
// -1 = none found
int find_peak_to_right(
    int proj[],          // The projection (an array)
    int proj_size,       // The max size of the projection (not inclusive)
    int pos,             // Where to start looking
    int min_peak_height) // Minimum value for the peak
{
    int peak;            // Where the peak is

    peak = -1;           // Set to not found
    while ((++pos < proj_size) && (peak == -1))
      if (proj[pos-1] >= proj[pos]) && (proj[pos-1] >= min_peak_height))
        peak = pos - 1;
    return peak;
}

// Subroutine to find next peak to the left in a projection
// -1 = none found
int find_peak_to_left(
    int proj[],          // The projection
    int proj_min,        // The lowest index value of proj[]
    int pos,             // Where to start looking
    int min_peak_height) // Minimum value for the peak
{
    int peak;            // Where the peak is

    peak = -1;           // Set to not found
    while ((--pos >= proj_min) && (peak == -1))
      if ((proj[pos-1] >= proj[pos]) && (proj[pos-1] >= min_peak_height))
        peak = pos - 1;
    return peak;
}

// Subroutine to find next valley to the right in a projection
// -1 = none found
int find_valley_to_right(
```

【図186】

```
int proj[],          // The projection
int proj_size,       // The max size of the projection (not inclusive)
int pos,             // Where to start looking
int max_valley_depth) // Highest point a valley can be
{
  int valley;        // Where the valley is

  valley = -1;       // Set to not found
  while ((--pos < proj_size) && (valley == -1))
    if ((proj[pos-1] <= proj[pos]) && (proj[pos-1] <= max_valley_depth))
      valley = pos - 1;
  return valley;
}

//////////////////////////////////////////////////////////////////////
// Subroutine to find next valley to the left in a projection
// -1 = none found
int i4_valley_to_left(
  int proj[],        // The projection
  int pos,           // The largest index value of proj[]
  int max_valley_depth) // Where to start looking
{
  int valley;        // Where the peak is

  valley = -1;       // Set to not found
  while ((--pos >= proj_min) && (valley == -1))
    if ((proj[pos] <= proj[pos-1]) && (proj[pos-1] <= max_valley_depth))
      valley = pos - 1;
  return valley;
}

//.....  Find the angles of the valley .....
// This routine will find the Peak-Valley-Peak (the
// P-V-P combo) of the histogram in VPP[]
void i4_find_PVP(
  int VPP_min,         // The min range of the VPP
  int VPP_max,         // The max range of the VPP
  int valley_pos,      // The position to the right if the 1st peak
  int min_peak_height, // Constants
  int max_valley_depth,// The max valley depth
  int *p1,             // The top of 1st peak
  int *v1,             // The start of valley
  int *v2,             // The end of valley
  int *p2)             // The top of 2nd peak
{
  // Find p1: the peak to the left of the valley
  *p1 = find_peak_to_left(VPP, min_peak_height);
  if (*p1 == -1)       // If none found, make it the left side
    *p1 = VPP_min;

  // Find v1: the valley to the right of the given peak
  *v1 = find_valley_to_right(VPP, VPP_max, *p1, max_valley_depth);
  if (*v1 == -1)       // If not found, make it the right side
    *v1 = VPP_max - 1;

  // Find p2: the next peak to the right of valley v1
  *p2 = find_peak_to_right(VPP, VPP_max, *v1, min_peak_height);
  if (*p2 == -1)       // If not found, make it the right side
    *p2 = VPP_max - 1;

  // Find v2: the nearest valley to the left of peak p2
  *v2 = find_valley_to_left(VPP, VPP_min, *p2, max_valley_depth);
  if (*v2 == -1)       // If not found, make it the left side
    *v2 = VPP_min;
}

//////////////////////////////////////////////////////////////////////
// This routine will find the angles given the P-V-P coordinates
// (The array is in VPP)
void i4_find_valley_angles(
  int p1,              // The top of 1st peak
  int v1,              // The start of valley
  int v2,              // The end of valley
  int p2,              // The top of 2nd peak
  int *theta1,         // Output: The index of angles
  int *theta2)         // ( * angle/ROT_DEG_RESOLUTION)
{
  int p1y;             // The height of left peak
  int v1y;             // The height of left valley portion
  int v2y;             // The height of right valley
  int p2y;             // The height of right peak
  double t1, t2;       // The output angle in radians

  p1y = VPP[p1];
  v1y = VPP[v1];       // Get its height:
  v2y = VPP[v2];
  p2y = VPP[p2];

  t1 = atan((double) (v1-p1)/ (double) (v1y-p1y)); // This is positive
  t2 = atan((double) (p2-v2)/ (double) (p2y-v2y)); // This is negative

  t1 = t1 * 180.0 / 3.14159 / ROT_DEG_RESOLUTION;
  t2 = t2 * 180.0 / 3.14159 / ROT_DEG_RESOLUTION;

  *theta1 = (int) (t1+.5);     // Round the angles
  *theta2 = (int) (t2-.5);
}

//.....  Find minimum in a rotated valley  .....
void i4_find_min_valley_at_theta(
  int image[],         // The image
  struct Character *ch,// The character to find the projection on
  int angle,           // The angle (scaled)
  int p1,              // The left range to look through
  int p2,              // The right range to look through
  int *min,            // The best minimum found
  int *min_at,         // It's location
  int *min_thresh)     // The angle where it's located
{
  int min, max;        // Min and max indices of Proj
  int max_value_proj;  // The maximum value of the projection
  int t1,t2;           // Range to search for minimum (inclusive)
  int it_height;       // The minimum height needed for a peak (be
                       //   left & right sides won't have a minimum)
  int i;

  if (*min == 0)       // If a zero found!
    return;            // Don't bother doing the rest

  // Find the projection (load into Proj[])
  i4_find_projection(image, ch, angle, p1, p2, Proj, &min, &max);

  max_value_proj = Proj[min];
  for (i=min+1; i<=max; i++)
    if (Proj[i] > max_value_proj)
      max_value_proj = Proj[i];

  // Choose range to ignore points before first peaks on left and right
  t1 = min;
```

【図187】

```
    >2 - peak-1;
    ir_height = (int) (((float) max_value_prj) * IA_IGNORE_WEIGHT+.5);
                                    // How high to ignore the ends

    // Find left peak (while slope is still rising), and above IA_IGNORE_WEIGHT
    while ((Proj[r2-1]) <= Proj[r2]) || (Proj[r2-2]) ||
           (Proj[r2] <= Proj[r2-3]) || (Proj[r2] < ir_height))
        if ((--r2) >= peak - 3)
                                    // If get to left edge
            return;

    // Find right peak (while slope is rising from right to left)
    while ((Proj[r2] <= Proj[r2-1]) || (Proj[r2-2]) ||
           (Proj[r2] <= Proj[r2-3]) || (Proj[r2] < ir_height))
    {
        r2++;
        if (r2 >= n) // || (r2 >peak+3))    // If reached the left point
            return;
    }

    // Search this range, and find best minimum
    for (i=r2; i<r2; i++)
    {
        if (Proj[i] <= *min)                // Found a better minimum
        {
            if (Proj[i] != *min)
            {
                *min = Proj[i];
                *min_at = i;
                *min_check = angle;
            } else {
            if (labs(angle) < labs(*min_check))
            {
                printf("tiebreaker broke.....\n");
                *min_at = i;
                *min_check = angle;
            }
        }
    }
}

//.......... The find optimal rotated histogram routines  ..........
////////////////////////////////////////////////////////////////////
// The insert angular break: Attempts to insert break in a character at a given
// thresh level.  This updates ch->num and ch->level_cut_at for both
// characters
int insert_angular_break(           // Returns number chars (1 or 0)
    unsigned image,
    struct CharList *clist,         // The character list to add chars to
    struct Character *ch,           // The character to be cut
    int thresh_level,               // The threshold levels
    int repeating_char)             // =1 if the last time this routine was called
                                    //    was on the same character, and no breaks
                                    //    have inserted (so we don't have to recompute
                                    //    the VPP, etc)
{
    static int VPP_min, VPP_max;    // The start and stop values of the VPP
    static int max_value_vpp;       // The maximum value of the VPP
    int max_valley_depth;           // Maximum height for a valley
    int min_peak_height;            // Minimum height for a peak
    int valley_pos;                 // Position of valley
    int thresh1, thresh2;           // Angles of valley walls (BEST_DEL_RESOLUTION)
    int p1,v1,v2,p2;                // The Peak/Valley/Peak column numbers
    int min_at;                     // The location of the minimum
    int min_check;                  // The angle of the minimum
    int min;                        // The value of the minimum
    int range_ch_min;               // Range of angles to search for in chars
    int range_ch_max;               //
    int range_ch_min2;              // 2nd range of angles (dito)
    int range_ch_max2;              //
    int angle;                      // What angle we're finding the projection for
    struct Character *another;      // The new character to be inserted
    int i,j,k,r,r2;

    if (repeating_char == 0)        // Remember last if recomputing
    {
        IA_find_projection(image, ch, 0, ch->left, ch->right, VPP, &VPP_min, &VPP_max);

        // Find maximum projection (for find peak and valley heights)
        max_value_VPP = VPP[VPP_min];
        for (i=VPP_min+1; i<VPP_max; i++)
            if (VPP[i] > max_value_VPP)
                max_value_VPP = VPP[i];
        } else {
        printf("** VPP Remembered: Max_value_VPP = %d\n", max_value_VPP);
    }

    // Compute max_valley_depth and min_peak_height
    max_valley_depth = (int) max_value_VPP * Const1[thresh_level];
    min_peak_height = (int) ((float) max_value_VPP * Const1[thresh_level]);
    if (max_valley_depth >= min_peak_height)
        min_peak_height++;

    // printf("max_value_VPP = %d\n", max_value_VPP);
    // printf("Threshlevel = %d\n", thresh_level);
    // printf("Const1,2 = %f, %f\n", Const1[thresh_level], Const2[thresh_level]);
    // printf("max_valley_depth = %d\n", max_valley_depth);
    // printf("min_peak_height = %d\n", min_peak_height);

    /*** BEGIN PLOTTING ***/
#ifdef VISIBLE
    int n;
    if (repeating_char == 0)
    {
        IA_plot_new_column(0);
        IA_plot_char(image, ch, CursorX, CursorY);      // Go to a new column
        CursorX += ch->box + 6;
        CursorY += max_value_VPP;
        for (n=VPP_min; n<VPP_max; n++)                 // Plot VPP
            XD_plot(CursorX + n - VPP_min, CursorY - (int) (((float) 25*VPP[n]) / max_value_VPP));
        if (CursorX < CursorX + VPP_max - VPP_min)      // Adjust next col loc
            CursorX = CursorX + VPP_max - VPP_min;
        Xd_flush();
        CursorX_VPP_first = CursorX;
        CursorY_VPP = CursorY;
        CursorX += 20;
    }
#endif
    /*** END PLOTTING ***/

    // Find first peak to right
    i = find_peak_to_right(VPP, VPP_max, VPP_min, min_peak_height);
    if (i == -1)                    // If peak not found...
        return 0;

    // Search for valley to right
    // Note: i should be the peak on the left!
    valley_pos = find_valley_to_right(VPP, VPP_max, i, max_valley_depth);
    if (valley_pos == -1)           // If not found
        return 0;
```

[図188]

```
  // Search for next peak to right of valley_pos
  i2 = find_peak_to_right(VPP, VPP_max, valley_pos, min_peak_height);
  if (i2 == -1)           // If not found
  return 0;

  // Now compute the angles
  i4_find_PVVP(VPP_min, VPP_max, valley_pos, min_peak_height,
                  max_valley_depth, &p1, &v1, &v2, &p2);

  /*** BEGIN PLOTTING ***/
#ifdef VISIBLE
  // Plot the tick mark under valley found
  for (c = p1; c <= p2; c++)
    xs_plotxy(Cursor_vpp_tickX+c-VPP_min, Cursor_vpp_tickY);
  xs_plotxy(Cursor_vpp_tickX+p1-VPP_min, Cursor_vpp_tickY-1);
  xs_plotxy(Cursor_vpp_tickX+p1-VPP_min, Cursor_vpp_tickY-3);
  xs_plotxy(Cursor_vpp_tickX+p2-VPP_min, Cursor_vpp_tickY-1);
  xs_plotxy(Cursor_vpp_tickX+p2-VPP_min, Cursor_vpp_tickY-3);
  xs_flush();
#endif
  /*** END PLOTTING ***/

  // If we've done this VPP before, get results.  If not, compute them
  if (i4_PVVP_cache_hit(p1,v1,v2,p2,ch,&min_theta,&min_at) == 0)
  {
    i4_find_valley_angles(p1, v1, v2, p2, &theta1, &theta2);
//    printf("Valley angles = %d, %d\n", theta1, theta2);

    range_t1_min = theta1-3;    // theta1 - 3,6,9
    range_t1_max = theta1+3;  // theta1 + 3,6,9
    range_t2_min = theta1-3;   // theta2 - 3,6,9
    range_t2_max = theta2+3;  // theta2 + 3,6,9

    // Check ranges
    if (range_t1_max > ROT_INDEX_RANGE)
      range_t1_max = ROT_INDEX_RANGE;
    if (range_t2_max > ROT_INDEX_RANGE)
      range_t2_max = ROT_INDEX_RANGE;
    if (range_t1_min < -ROT_INDEX_RANGE)
      range_t1_min = -ROT_INDEX_RANGE;
    if (range_t2_min < -ROT_INDEX_RANGE)
      range_t2_min = -ROT_INDEX_RANGE;

    if (range_t1_max >= range_t1_min)  // If overlap :)
    {
      printf(" Range reduced: (%d:%d), (%d:%d) to ", range_t1_min,
        range_t1_max, range_t2_min, range_t2_max);
      range_t1_max = range_t2_max;              // Then combine & do one search
      range_t2_min = range_t2_max + 1;
      printf("(%d:%d)\n", range_t1_min, range_t1_max);
    }

    // Search the angle range and find the best breakbreak the image
    min = VPP[v2];                      // Search 0 degrees
    min_at = v2;
    min_theta = 0;
    for (j=v1; j<v2; j++)
      if (VPP[j] < min)
      {
        min = VPP[j];
        min_at = j;
      }
    // Search rest of them
    for (angle = range_t1_min; angle <= range_t1_max; angle++)
      i4_find_min_valley_at_theta(image, ch, angle, p1, p2,
                                  &min, &min_at, &min_theta);

    for (angle = range_t1_min; angle <= range_t2_max; angle++)
      i4_find_min_valley_at_theta(image, ch, angle, p1, p2,
                                  &min, &min_at, &min_theta);

    i4_PVVP_cache_insert(p1,v1,v2,p2,ch,min_theta,min_at);
                                        // Place in cache
//    printf("Minimum found: Angle = %d, at %d = %d\n", min_theta, min_at, min);
  } else {
    printf(" Cache Hit!\n");
//    printf("Cache: Angle = %d, at %d\n", min_theta, min_at);
  }

  /*** BEGIN PLOTTING ***/
#ifdef VISIBLE
  char string[100];
  int pmin, pmax, max_value_proj, tx, ty, tmin;
  if (Cursor4_Y > WINDOW_SIZE_Y - 50)           // Wrap over to new column
    i4_plot_new_column(1);
  sprintf(string, "Lev %d", thresh_level);      // Plot level
  xs_string(Cursor4_X, Cursor4_Y, string);
  Cursor4_Y += 8+3;
  sprintf(string, "Deg %d", min_theta*3);       // Plot angle
  xs_string(Cursor4_X, Cursor4_Y, string);
  Cursor4_Y += 8+1;

  // Load rotated projection into array again
  i4_find_projection(image, ch, min_theta, p1, p2, Proj, &pmin, &pmax);
  max_value_proj = Proj[pmin];          // Compute height of Proj[]
  for (i=pmin+1; i<pmax; i++)
    if (Proj[i] > max_value_proj)
      max_value_proj = Proj[i];
  Cursor4_Y += max_value_proj;          // Plot Proj[]
  for (c=pmin, c=pmax; c++)
    for (r=0; r<=Proj[c]; r++)
      xs_plotxy(Cursor4_X+c-pmin, Cursor4_Y-r);
  if (Cursor3_X < Cursor4_X + pmax - pmin)      // Adjust next col loc
    Cursor3_X = Cursor4_X + pmax - pmin;
  c = Cursor4_X + min_at - pmin;
  r = Cursor4_Y+2;
  xs_plotxy(c, r);                              // Plot Tickmark
  xs_plotxy(c-1, r+1); xs_plotxy(c, r+1); xs_plotxy(c+1, r+1);
  xs_plotxy(c-2, r+2); xs_plotxy(c-1, r+2); xs_plotxy(c, r+2);
  xs_plotxy(c+1, r+2); xs_plotxy(c+2, r+2);
  Cursor4_Y += 10;

  tx = Cursor4_X; ty = Cursor4_Y;       // Plot character (tx,ty=pos)
  if (Cursor3_X < Cursor4_X + ch->right - ch->left)     // Adjust next col loc
    Cursor3_X = Cursor4_X + ch->right - ch->left;
  i4_plot_char(image, ch, tx, ty);
  Cursor4_Y += ch->bot - ch->top + 9;

  if (min_theta > 0)                    // Plot cut line
  {
    tmin = min_theta-1;
    for (r=ch->top-6; r<ch->bot; r++)
      xs_plotxy(tx+Rot[tmin][ch->bot-r-1]+min_at-ch->left,ty+r);
  }
  if (min_theta < 0)                    // Plot cut line
  {
    tmin = -min_theta-1;
    for (r=ch->top-6; r<ch->bot; r++)
      xs_plotxy(tx+Rot[tmin][ch->bot-r-1]+min_at-ch->left,ty+r);
  }
  if (min_theta == 0)                   // Plot cut line
  {
    for (r=ch->top-6; r<ch->bot; r++)
      xs_plotxy(tx+min_at-ch->left,ty+r);
```

```
    }

    Cursord_Y += 20;                            // Move cursor down

    xs_flush();
  !endif
    /*** END PLOTTING ***/


    // Break the Character
    newchar = &(clist->chr[clist->size]);  // Set newchar to point to the end
                                           // of the charlist (the next blank character)

    if (min_theta == 0)           // Horizontal Break
    {
      for (r=ch->top; r<ch->bot; r++)
      {
        newchar->curve1[r] = ch->curve1[r];       // Break the character
        b = min_at;
        if (b < ch->curve1[r])
          b = ch->curve1[r];
        if (b > ch->curve2[r])
          b = ch->curve2[r];
        newchar->curve2[r] = ch->curve2[r] = b;
      }
      newchar->right = ch->right;
      ch->right = min_at;
      newchar->left = min_at;
    } else {
      if (min_theta > 0)  // Positive angled break
      {
        t = min_theta-1;  // The index to use for the images
        for (r = ch->top; r<ch->bot; r++)
        {
          newchar->curve1[r] = ch->curve1[r];
          b = min_at + Rot[t][ch->bot-r-1];
          if (b < ch->curve1[r])
            b = ch->curve1[r];
          if (b > ch->curve2[r])
            b = ch->curve2[r];
          newchar->curve2[r] = ch->curve2[r] = b;
        }
        newchar->right = ch->right;
        ch->right = min_at + Rot[t][ch->bot-1];
        newchar->left = min_at;
      } else {                    // Negative angled break
        t = -min_theta-1;
        for (r=ch->top; r<ch->bot; r++)
        {
          newchar->curve2[r] = ch->curve2[r];
          b = min_at - Rot[t][ch->bot-r-1];
          if (b < ch->curve1[r])
            b = ch->curve1[r];
          if (b > ch->curve2[r])
            b = ch->curve2[r];
          newchar->curve1[r] = ch->curve1[r] = b;
        }
        newchar->right = ch->right;
        ch->right = min_at;
        newchar->left = min_at - Rot[t][ch->bot-1];

      }
    }
    newchar->top = ch->top;
    newchar->bot = ch->bot;
    newchar->level_cut_at = thresh_level-1;
    ch->level_cut_at = thresh_level;

    newchar->rec = UNRECOGNIZED;
    ch->rec = UNRECOGNIZED;

    l4_char_reset_lr(image, ch, newchar); // Update left & right borders
    Charlist_insert_after_cut(clist, ch);

    return 1;
  }


  //**********************************************************/
  //**********************************************************/
  //              THE GOODCUT ROUTINES SECTION
  //**********************************************************/
  //**********************************************************/

  ////////////////////////////////////////////////////////////////////////
  // This routine will cut a character, starting at level 0, and going up
  // Through the levels until it makes a break.  Returns 1 if successful.
  // 0 if unsuccessful.
  int l4_cut_character(
    cimage& image,
    struct Charlist *clist,       // The character list to add chars to
    struct Character *ch)         // The character to be cut
  {
    int level;                    // The level of the break
    int repeat;                   // If repeating

    repeat = 0;                   // 1st call is not repeating
    for (level = 0; level < max_thresh_level; level++)    // Loop through levels
    {
      if (insert_angular_break(image, clist, ch, level, repeat))
        return 1;
      repeat = 1;                 // We're repeating from now on
    }

    return 0;                     // No break inserted
  }


  ////////////////////////////////////////////////////////////////////////
  void l4_combine_chars(
      struct Charlist *clist,
      struct Character *ch1,      // The character to start from
      struct Character *ch2)      // The character to start from
  {
    struct Character *c;          // Looping character
    int i;
    if (ch1 == ch2)               // Return if equal
      return;
    if ((ch1->top != ch2->top) || (ch1->bot != ch2->bot))
    {
      printf("Error in l4_combine_chars||: Combing characters who'se top's\n");
      printf("and bottoms are not equal: (t,b) = (%d,%d), (%d,%d)\n",
          ch1->top, ch1->bot, ch2->top, ch2->bot);
      exit(1);
    }

    for (i=ch1->top; i<ch1->bot; i++)
      ch1->curve2[i] = ch2->curve2[i];
    ch1->right = ch2->right;

    // Remove all the characters between the two
    c = ch2;
    while(c != ch1)
    {
```

```
    Charlist_remove_char(clist, c);
    c = c->prev;
  }
  Charlist_recalculate_loc(ch);        // Recalculate location
}


//////////////////////////////////////////////////////////////////////////
// This routine works with 14_goodcut_comb_left() to check '99''s after
// successful recognition.  Ie.  N N(99) Y  tries N [N(99) Y]  which
// will detect and remove some errors caused by an false successful recognition
int 14_goodcut_99_left(
    cimage& image,
    struct Charlist *clist,
    struct Character *ch,        // The character to start from
    struct Character *lch)       // Leftmost character to search (exclusive)
{
  struct Character *c;           // The leftmost character of the range
  int ret_value;                 // The # of chars removed
  int temp_value;

  ret_value = temp_value = 0;
  c = ch;

  while(1)
  {
    c=c->prev;                   // Get the previous character
    temp_value++;
    if (c == lch)                // At edge
      return ret_value;
    if (c->rec != FAILED_RECOGNITION)   // Return if not a N
      return ret_value;
    if (c->level_cut_at != max_thresh_level)    // Return if not '99'
      return ret_value;
    if (recognition(image,c,ch))
    {
      14_combine_chars(clist, c,ch);    // Combine it into one character (in c)
      c->rec = PASSED_RECOGNITION;
      ret_value += temp_value;
      temp_value = 0;
      ch = c;
    }
  }
}


//////////////////////////////////////////////////////////////////////////
// This routine will start at character ch and search to left from ch
// grouping characters to see if they're recognized.  If so they are
// combined.  Note: lch can == 0.
14_goodcut_comb_left(
    cimage& image,
    struct Charlist *clist,
    struct Character *ch,        // The character to start from
    struct Character *lch)       // Leftmost character to search (exclusive)
{
  struct Character *c;           // The leftmost character of the range
  int ret_value;                 // The # of chars removed

  ret_value = 0;
  c = ch;                        // Start at character we're on

  while(1)                       // This loop is 'return'ed out of
  {
    if (recognition(image, c, ch))  // c to ch recognized?
    {
      14_combine_chars(clist, c,ch);    // Combine it into one character (in c)
      c->rec = PASSED_RECOGNITION;
      ret_value += 14_goodcut_99_left(image,clist,c,lch);  // Do the 99's
      return ret_value;
    }
    if (c == ch)
      c->rec = FAILED_RECOGNITION;
    c = c->prev;                 // Incorporate next character to left
    ret_value--;

    // Check for exits
    if (c == lch)                // If we're at the end, exit
      return 0;
    if (c->next->rec == PASSED_RECOGNITION)     // If we've passed a Y
      return 0;                  // Stop searching
    if (14_too_long(c,ch))       // If character is too long
      return 0;
  }
}


//////////////////////////////////////////////////////////////////////////
// This routine works with 14_goodcut_comb_right() to check '99''s after
// successful recognition, or when a top level is hit.
//    Ie.  N N(99) ch  tries N [N(99) ch]  which
// will detect and remove some errors caused by an false successful recognition
// and by not checking adjacent characters when can't split at level 99
int 14_goodcut_99_right(
    cimage& image,
    struct Charlist *clist,
    struct Character *ch,        // The character to start from
    struct Character *rch)       // Rightmost character to search (exclusive)
{
  struct Character *c;           // The rightmost character of the range
  int ret_value;                 // The # of chars removed
  int temp_value;

  ret_value = temp_value = 0;
  c = ch;

  while(1)
  {
    c=c->next;                   // Get the previous character
    temp_value++;
    if (c == rch)                // At edge
      return ret_value;
    if (c->rec != FAILED_RECOGNITION)   // Return if not a N
      return ret_value;
    if (c->level_cut_at != max_thresh_level)    // Return if not '99'
      return ret_value;
    if (recognition(image,ch,c))
    {
      14_combine_chars(clist, ch,c);    // Combine it into one character (in c)
      ch->rec = PASSED_RECOGNITION;
      ret_value += temp_value;
      temp_value = 0;
      c = ch;
    }
  }
}


//////////////////////////////////////////////////////////////////////////
// This routine will start at character ch and search to right from ch
// grouping characters to see if they're recognized.  If so they are
// combined.  Note: lch can == 0.
14_goodcut_comb_right(
    cimage& image,
```

【図191】

```
    struct Charlist *clist,
    struct Character *ch,        // The character to start from
    struct Character *rch)       // Rightmost character to search (exclusive)
{
  struct Character *c;           // The rightmost character of the range
  int ret_value;                 // The # of chars removed

  ret_value = 1;
  c = ch;                        // Start at character before ch

  while(1)                       // This loop is 'return'ed out of
  {
    if (recognition(image, ch, c))  // ch to c recognized?
    {
      l4_combine_chars(clist, ch,c);  // Combine it into one character (in ch)
      ch->rec = PASSED_RECOGNITION;
      ret_value += l4_goodcut_99_right(image,clist,ch,rch);  // Do the 99'ec
      return ret_value;
    }
    if (ch == c)
      ch->rec = FAILED_RECOGNITION;
    c = c->next;                 // Incorporate next character to right
    ret_value++;

    // Check for exits
    if (c == rch)                // If we're at the end, exit
      return 0;
    if (c->prev->rec == PASSED_RECOGNITION)    // If we've passed a Y
      return 0;                  // Stop searching
    if (l4_too_long(ch,c))       // If character is too long
      return 0;
  }
}

//**************************************************************/
// The goodcut routine
//**************************************************************
// This will run goodcut on a character to be broken up
int goodcut_break_character(     // Returns number of chars added
    cimage4 image,
    struct Charlist *clist,
    struct Character *chin)
{
  struct Character *left_border;        // The left border character
  struct Character *right_border;       // The left border character
  struct Character *cut;                // The character to cut
  struct Character *ch;                 // A looping variable
  int chars_cut;                        // The number of characters added
  int lowest_level;                     // The level of the character to cut
  int a;

  chars_cut = 0;
  left_border = chin->prev;
  right_border = chin->next;

  while(1)                              // Loop until broken out of
  {
    lowest_level = max_thresh_level;
    ch = chin;                          // Get 1st character
    while (ch != right_border)          // Loop until searched all characters
    {
      if (ch->rec == FAILED_RECOGNITION)      // If N
        if (ch->level_cut_at < lowest_level);  // If was cut at a lower level
        {
          lowest_level = ch->level_cut_at;    // then cut this char next
          cut = ch;
        }
      ch = ch->next;
    }
    if (lowest_level == max_thresh_level) // Exit if no character to cut
      break;
//  print_asc_charlist(clist,cut);    // ******* DELETE ME *******
    if (l4_cut_character(image,clist,cut) == 0) // Break the character
    {
      cut->level_cut_at = max_thresh_level;     // Can't break character
    } else {
      chars_cut++;
      // Try to combine the characters
      ch = cut->next;   // Remembered in case next line removes char "cut"
      chars_cut -= l4_goodcut_comb_left(image,clist,cut,left_border);
      chars_cut -= l4_goodcut_comb_right(image,clist,ch,right_border);
    }
  }

  return chars_cut;
}

//**************************************************************/
//  This passes each character to the goodcut routine
int goodcut(                  // Returns number of charactes added
    cimage4 image,
    struct Charlist *clist)
{
  int chars_added;        // The number of characters added (output)
  struct Character *ch; // Used when looping (the beggining of the chunck)
  struct Character *next_ch; // Temp storage character (where to continue)

  chars_added = 0;
  // Set all break levels to -1 and
  // pass all the unrecognized characters to recognition
  ch = clist->first;
  while(ch != 0)
  {
    next_ch = ch->next;
    ch->level_cut_at = -1;
//    if (ch->rec == UNRECOGNIZED)       //**** FORCE A NOT RECOGNIZED ON THEM
      if (recognition(image, ch, ch))
        ch->rec = PASSED_RECOGNITION;
      else
        ch->rec = FAILED_RECOGNITION;
    if (ch->rec == FAILED_RECOGNITION)
      chars_added += goodcut_break_character(image, clist, ch);
    ch = next_ch;
  }
  printf("characters added = %d\n",chars_added);
  return chars_added;            // Exit
}

//**************************************************************
//**************************************************************
//*************                  MAIN              *************
//**************************************************************
//**************************************************************
//////////////////////////////////////////////////////////////////
// Initialize:
// Should be called prior to using.  This calles all the
// initialization routines
void l4_initialize()
{
  l4_initialize_ret();
  l4_initialize_constants();
```

【図192】

```
}

//////////////////////////////////////////////////////////////////
//
void level4(
  cimage4 line,                 // The line
  struct charlist *clist)       // The charlist
{
  int proj_min, proj_max,       // Used to compute size of proj
  int i;

#ifdef VISIBLE
  xs_clr_win();
  int c;
  Cursor1_X = Cursor1_Y = 10;
  line.plot(Cursor1_X, Cursor1_Y);    // Plot input image
  Cursor2_Y = Cursor1_Y + line.sizeR + 30;      // Final output goes at Cursor2
  Cursor2_X = Cursor1_X;
  Cursor3_Y = Cursor2_Y + line.sizeR + 30;
  Cursor3_X = Cursor2_X;
  for (c=0; c<WINDOW_SIZE_X; c++)
    xs_plotxy(c, Cursor3_Y);            // Plot horiz line
  Cursor3_Y = Cursor3_Y + 20;
  Cursor4_Y = Cursor3_Y;
  Cursor4_X = Cursor3_X;
#endif

  if (line.sizeR > MAX_CHAR_HEIGHT)
  {
    printf("\n*** ERROR ***\n");
    printf("*** Line size (%d) is greater than max line size allocated in multilevel
\n", line.sizeR);
    printf("*** routines (MAX_CHAR_HEIGHT = %d).  Increase MAX_CHAR_HEIGHT.\n", MAX_C
HAR_HEIGHT);
    exit(1);
  }

  // Allocate Proj and VPP
  i = (ROT_MAX_RANGE/ROT_DEG_RESOLUTION)-1;
  proj_min = -Roc[i][line.sizeR];
  proj_max = line.sizeC-proj_min;
  Proj = ivector_ranged(proj_min, proj_max);
  VPP = ivector(line.sizeC);
  FVVP_zero = 1;                        // Erase the cache

  /*  ..... Program goes here ..... */
  goodcut(line, clist);

#ifdef VISIBLE
  // Print final character list
  i4_plot_charlist(line, clist, Cursor2_X, Cursor2_Y);
#endif
  printf("Displaying final level 4 charlist.  Enter 0 to continue? ");
//  print_rec_charlist(clist,clist->first);     // ******* DELETE ME ********
  i = auto_input_int();

  // Deallocate
  free_ivector(VPP);
  free_ivector_ranged(Proj, proj_min);
}
```

【図203】

```
/*
 * The routines in this file determine if a character is recognized or not
 * (for segmentation)
 *
 *.....
 */
#include"xs.h"                  /* Include Rick's graphics routines */
#include"autoinput.h"

extern int WINDOW_SIZE_Y;

int recognized(a, size_r, size_c) /* Return a 0 for not recognized, 1 for
                                     recognized */
     int **a;                   /* The character matrix a[row][col] */
     int size_r;                /* The number of rows in a */
     int size_c;                /* The number of columns in a */
{
  int r,c;                      /* The row and column of the image */

  int dummy;                    /* A dummy variable */


#ifdef VISIBLE
  /* This routine will print the char at the screen's bottom left corner */
  for (r=0-10; r<size_r+20; r++)        /* Loop through the rows */
    for (c=0-10; c<size_c+20; c++)      /* Loop through the cols */
      if ((r >= 0) && (r < size_r) && (c >= 0) && (c < size_c))
        if (a[r][c] != 0)               /* If the pixel is on */
          xs_plotxy(c+10, r + WINDOW_SIZE_Y - size_r - 20);
        else
          xs_unplotxy(c+10, r + WINDOW_SIZE_Y - size_r - 20);
      else
        xs_unplotxy(c+10, r + WINDOW_SIZE_Y - size_r - 20);

  xs_flush();                   /* Flush the display buffer (to make sure
                                   the character gets printed on the screen */
#endif

  printf("Character displayed.  Enter return value (1 = good, 0 = bad)? ");
  dummy = auto_input_int();
  return dummy;
}
```

【図193】

```
/* This program contains common vector and matrix mallocs */
#include<malloc.h>
#include<stdio.h>

/* Routine for error during memory allocation */
void allocation_error(s)
     char* s;                    /* The error text */
{
  int *coredump;

  printf("Memory allocation error %s\nProgram Aborted.\n", s);
  *coredump = 0;
  coredump = 0;
}


/************ IVECTOR ***************/
/* Allocate an integer vector of size [0..n-1] */
int *ivector(n)
     int n;                      /* Size of vector */
{
  int *v;

  v = (int *)malloc((unsigned) n*sizeof(int));
  if (!v) allocation_error("in ivector()");
  return v;
}

/* Free an integer vector v of size [0..n-1] */
void free_ivector(v)
     int *v;                     /* The vector */
{
  free((int *) v);
}


/************ FVECTOR ***************/
/* Allocate an float vector of size [0..n-1] */
float *fvector(n)
     int n;                      /* Size of vector */
{
  float *v;

  v = (float *)malloc((unsigned) n*sizeof(float));
  if (!v) allocation_error("in fvector()");
  return v;
}

/* Free an float vector v of size [0..n-1] */
void free_fvector(v)
     float *v;                   /* The vector */
{
  free((float *) v);
}


/************ RANGED IVECTOR ***************/
/* Allocate a ranged vector v of size [m..n-1] */
int *ivector_ranged(m,n)
     int m,n;                    /* The range */
{
  int *v;

  v = (int *)malloc((unsigned) (n-m)*sizeof(int));
  if (!v) allocation_error("in ivector_ranged()");
  return (v-m);
}


/* Free an integer ranged vector v of size [m..n-1] */
void free_ivector_ranged(v,m)
     int *v;                     /* The vector */
     int m;                      /* The starting index of v */
{
  free((int *) (v+m));
}


/************ IMATRIX ***************/
/* Allocate an integer matrix of size [0..r-1][0..c-1] */
int **imatrix(r, c)
     int r,c;                    /* The size of the matrix */
{
  int** m;
  int i;

  m=(int **)malloc((unsigned) r * sizeof(int *));
  if (!m) allocation_error("1 in imatrix()");
  for (i=0; i<r; i++) {
    m[i] = (int *)malloc((unsigned) c * sizeof(int));
    if (!m[i]) allocation_error("2 in imatrix()");
  }
  return m;
}

/* Free an integer matrix m of size [0..r-1][0..c-1] */
void free_imatrix(m, r)
     int **m;                    /* The matrix */
     int r;                      /* The size (num rows) */
{
  int i;

  for (i=r-1; i>=0; i--)
    free((int *) m[i]);
  free((int **) m);
}


/************ CMATRIX ***************/
/* Allocate an char matrix of size [0..r-1][0..c-1] */
char **cmatrix(r, c)
     int r,c;                    /* The size of the matrix */
{
  char** m;
  int i;

  m=(char **)malloc((unsigned) r * sizeof(char *));
  if (!m) allocation_error("1 in cmatrix()");
  for (i=0; i<r; i++) {
    m[i] = (char *)malloc((unsigned) c * sizeof(char));
    if (!m[i]) allocation_error("2 in cmatrix()");
  }
  return m;
}

/* Free an character matrix m of size [0..r-1][0..c-1] */
void free_cmatrix(m, r)
     char **m;                   /* The matrix */
     int r;                      /* The size (num rows) */
{
  int i;

  for (i=r-1; i>=0; i--)
    free((char *) m[i]);
```

【図196】

```
//
// Filename: outline.c
// Outlining routines, for character segmentation
//
/////////////////////////////////////////////////////////////////
// Bits used in the cimage:
// Bit 0: (1)    : 0 = Pixel off, 1 = pixel on
// Bit 1: (2)    : 1 = Part of an outline
// Bit 2: (4)    : 1 = Has been extracted for outlined (don't go & outline it!)
// Bit 3: (8)
// Bit 4: (16)
// Bit 5: (32)
// Bit 6: (64)
// Bit 7: (128) -

#include"charlist.h"
#include"cimages.h"

#define VERY_SMALL -32000       // A very small integer (for image size)
#define VERY_LARGE 32000        // A very large integer
                                // considered for overlapping case.
// Define Directions used by the cursor for outlining
#define LEFT  0
#define UP    1
#define RIGHT 2
#define DOWN  3

// Define global variables for this program.  Used by programs on and
// below find_break_from_outline() routines.  t stands for turtle
int t_start;              // The left boundry column (inclusive)
int t_stop;               // The right boundry column (exclusive)
int t_cursorR;            // The row and column of the cursor for outlining
int t_cursorC;
int t_cursorD;            // The direction of the cursor

// Define external variables and routines (for displaying and debugging)
extern int cursor_X;
extern int cursor_Y;
extern print_character(cimage& line, struct Character *chr, int x, int y);
extern print_charlist(cimage& line, struct Charlist *clist, int x, int y);


// Test routine: Prints a cimage with outlining and restores image
void print_outlining(cimage& image, int s)
{
  int a,b;

  a = s-75; b = s+75;
  if (a < 0)  a = 0;
  if (b > image.sizeC)  b = image.sizeC;

  printf("\n");
  printf("Column Range = %d to %d\n", a, b);
  for (int r=0; r<image.sizeR; r++)
  {
    for (int c=a; c<b; c++)
    {
      switch (image.pixel[r][c])
      {
        case 0:
          printf(".");
          break;
        case 1:
          printf("o");
          break;
        case 3:
          printf("O");
          break;
        case 4:
          printf(",");
          break;
        case 5:
          printf("*");
          break;
        case 7:
          printf("#");
          break;
        case 123:
          printf("&");
          image.pixel[r][c] = 1;
          break;
        default:
          printf("?");
          break;
      }
    }
    printf("\n");
  }
  printf("\n");
}


//*******************************************************************//
//*******************************************************************//
//*******************************************************************//
//
//                    THE NEW ROUTINES.   (Delete all of the above)
//
//*******************************************************************//
//*******************************************************************//
//*******************************************************************//

/////////////////////////////////////////////////////////////////
//////                  The Outlining Routines               //////
/////////////////////////////////////////////////////////////////

/////////////////////////////////////////////////////////////////
// Turn left and right inline expansions
inline void t_turn_left() { t_cursorD = (t_cursorD-1) & 3;}
inline void t_turn_right() { t_cursorD = (t_cursorD+1) & 3;}

/////////////////////////////////////////////////////////////////
// This routine will move the mouse in the direction it is facing
void t_move_forward()
{
  switch(t_cursorD)
    {
      case LEFT:
        t_cursorC--;
        break;
      case RIGHT:
        t_cursorC++;
        break;
      case UP:
        t_cursorR--;
        break;
      case DOWN:
        t_cursorR++;
        break;
    }
```

【図197】

```
}

////////////////////////////////////////////////////////////////////
// This routine will return what's infront of where the mouse is facing
// It returns a 0 if it is out of range
int t_in_front(image* image)
{
    int oldR, oldC;                 // Stores the old direction
    int R;                          // What to return
    oldR = t_cursorR;               // Remembers this position
    oldC = t_cursorC;

    t_move_forward();               // Move into the new space
    // Check range
    if ((t_cursorR < 0) || (t_cursorR >= image->ysize))
    {
        t_cursorR = oldR;           // Restore cursor position
        t_cursorC = oldC;
        return 0;                   // It too high or low
    }
    if ((t_cursorC < t_start) ||
        (t_cursorC >= t_stop))
    {
        t_cursorR = oldR;           // Restore cursor position
        t_cursorC = oldC;
        return 0;                   // It to the left or right of the boarder
    }
    p = image->pixel[t_cursorR][t_cursorC];
    t_cursorR = oldR;
    t_cursorC = oldC;
    return p;
}

////////////////////////////////////////////////////////////////////
// Note: This routine won't outline 1 pixel if only one is set.
void extract_outline_char(
    image* line,                    // Image of the line
    struct charlist* clist,         // The character list
    int left_boundry,               // The left boundry of the area to outline
    int right_boundry,              // The right boundry of the area to outline
    int r,                          // The row and column of an 'on' pixel
    int c)                          // On the edge
{
    struct Character *new_char;     // Where the new character is going
    int top, bot, left, right;      // Used to be put in the new character
    int curve1, curve2;             // Pointers to curve1 and curve2 of new char.
    int startingR, startingC;       // The starting condition
    int not_finished;               // Curves when finished looping
    int i;

    // Set up convient pointers to the new character we're inserting
    new_char = &(clist->char[clist->site]); // Get new_char to the last character
                                    // on the character list.
    curve1 = new_char->curve1;      // Load in addresses of curves 1 & 2
    curve2 = new_char->curve2;

    // Initialize variables of new outlined character
    for (i=0; i<clist->vector_height; i++) // Initalize curve arrays
    {
        curve1[i] = VERY_LARGE;
        curve2[i] = VERY_SMALL;
    }
    top = bot = r;
    left = right = c;               // Max height
                                    // MAX left AND right increase

    // Initialize turtle stuff
    startingR = t_cursorR = r;      // Initalize cursor
    startingC = t_cursorC = c;
    t_cursorD = LEFT;               // Facing left
    t_start = left_boundry;
    t_stop = right_boundry;
    not_finished = 1;
    while(not_finished) // While we haven't returned to starting
    {
        // Adjust curves for this new position
        line.pixel[t_cursorR][t_cursorC] = line.pixel[t_cursorR][t_cursorC] | 2;
                                    // Set second bit to 1 (outlined)
        if (t_cursorC < curve1[t_cursorR]) // If turtle is left of current break
        {
            curve1[t_cursorR] = t_cursorC;      // Move break
            if (t_cursorC < left)               // Update left
                left = t_cursorC;
            if (t_cursorR < top)                // Update top and bot
                top = t_cursorR;
            if (t_cursorR > bot)
                bot = t_cursorR;
        }
        if (t_cursorC > curve2[t_cursorR]-1)    // (Same for right side)
        {   // Note: No need to update min
            curve2[t_cursorR] = t_cursorC+1;    //   and max because the left
                                                //   side (above) will get it
            if (t_cursorC > right)              // Update right
                right = t_cursorC;
        }

        // Check to see if we've entirely circled the image
        if ((t_cursorR == startingR) &&
            (t_cursorC == startingC))
                // We've returned to the starting point
        {
            i = t_cursorD;                  // Remember direction
            t_cursorD = UP;                 // Check the pixel above
            if ((t_in_front(line) & 7) != 1)    // If space above is on, and hasn't
            {                                   //   been outlined, don't exit)
                t_cursorD = LEFT;           // Check space left
                if ((t_in_front(line) & 7) != 1) // (Likewise for the left space)
                {
                    not_finished = 0;       // We're finished, exit the while routine
                    break;
                }
                t_cursorD = i;              // Restore direction and continue
            }

            // Move mouse to next position along the edge
            t_turn_left();                  // Check area to left of mouse
            if ((t_in_front(line) & 1) == 1)    // If something there
            {
                t_move_forward();           // Move to left
                continue;                   // And loop again
            }
            t_turn_right();
            t_turn_right();                 // Check area to right of mouse
            if ((t_in_front(line) & 1) == 1)    // If something there
            {
                t_move_forward();           // Move to right
                continue;                   // And loop again
            }
```

【図198】

```
        t_turn_right();             // Check area to behind of mouse
        if ((t_in_front(line) & 5) == 1)    // If something there
        {
          t_move_forward();         // Move behind mouse
          continue;                 // And loop again
        }
        not_finished = 0;           // Else it's a one-pixel image.  so exit
        line.pixel[t_cursorR][t_cursorC] = line.pixel[t_cursorR][t_cursorC] | 4;
                                    // Make this spot!
      }

      // Load the info into the Charlist
      new_char->top = top;
      new_char->bot = bot+1;        // (Make it exclusive)
      new_char->left = left;
      new_char->right = right+1;    // (Make it exclusive)
      new_char->type = CHARACTER_FULR_OUTLINED;

      Charlist_place_new_char(clist);       // Insert the new character

      // Mark the character as removed from the list
      for (r=0; r<line.sizeR; r++)
        for (c=curve1[r]; c < curve2[r]; c++)
          line.pixel[r][c] = line.pixel[r][c] | 4;
    }


    ////////////////////////////////////////////////////////////////////
    //////    Main routine to call to add breaks to the image    ///////
    ////////////////////////////////////////////////////////////////////
    void do_outlining(              // Returns number of characters added
        cimage& line,               // Image of the line
        struct Charlist* clist,     // The character list
        int start_col,              // The range to search through (inclusive)
        int stop_col)               //    (exclusive)
    {
      int r,c;                      // The row and column we're on

      for (c=stop_col-1; c>=start_col; c--) // Search through column range
        for (r=line.sizeR-1; r>=0; r--)     // Scan from bottom to top
          if (((line.pixel[r][c]) & 8) == 1)        // On pixel that hasn't been outlined
          {
//            line.pixel[r][c] = 12;
//            print_outlining(line, c);
            extract_outline_char(line, clist, start_col, stop_col, r, c);
                                    // Outline the character
          }
    }
```

【図208】

```
//
// Filename: test_mult.C
// This file is used to test and develop multilevel2.C
////////////////////////////////////////////////////////////////////
//
#include <stdio.h>
#include <string.h>
#include "cimages.h"
#include "mymalloc.h"
#include "XS.h"
#include "charlist.h"

extern void l4_initialize();
void level4(cimage& line, struct Charlist *clist);

void xs_init()
{
  struct Charlist clist;
  struct Character *ch;
  cimage image;
  int i,j;

  l4_initialize();

  Charlist_initialize(&clist, 300, 300);        // 10 chars, 100-max height

  char filename1[100];
  char filename2[100];
  printf("Input filename? /d2/tif/");
  scanf("%s", filename1);
  strcpy(filename2, "/d2/tif/");
  strcat(filename2, filename1);
  image.readtiff(filename2);
  image.clipspace();

  ch = &(clist.chr[clist.size]); // Set new_char to the last character

  // Load the character
  ch->top = 0;
  ch->bot = image.sizeR;
  ch->left = 0;
  ch->right = image.sizeC;
  for (i=ch->top; i<ch->bot; i++) {
    ch->curve1[i] = ch->left;
    ch->curve2[i] = ch->right;
  }

  Charlist_place_new_char(&clist);

  level4(image, &clist);

  printf("*** End of Program ***\n");
}
```

【図199】

```
//
// Filename: program.c
// This routine contains stuff to do page segmentation
//
//////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////
//.....
//

#define PASS_TO_LEVEL_4

#define MAX_CHARS_PER_LINE 250
#define MAX_LINE_HEIGHT 100

#include<stdlib.h>
#include<string.h>
#include"cimages.h"
#include"nu.h"
#include"myalloc.h"
#include"charlist.h"
#include"autoinput.h"

void process_line(cimage& image, struct Charlist *clist);
void segp_init(cimage page);
void segp_plot_line(cimage para, cimage line, Charlist *clist, int line_top,
                    int line_bot);

int segp_plot_page(cimage& page, int start_row);

#define NDIVISIONS 1    // The number of columns to divide the page into  (Original = 15)
#define CLOSE_OVERLAP 0.3    // What fraction of two lines must overlap (Original = 0.3)

#define PEDS_OFF_X_MIN_BLOCK_SIZE 0.024      // Minimum line height (inches)

#define RES (180.0/3.14159)          // Radians to Degrees
#define OVERLAP 1                    // Half the width of the overlap

// Global variables for page size calculations.  The line being segmented is not recalculated
extern cimage Process_line;

int PAGE_LL;                         // The starting row of the page (inclusive)
int PAGE_LR;                         // The stopping row of the page (exclusive)
int PAGE_UL = PAGE_LL;
int PAGE_CL;                         // The starting col of the page (inclusive)
int PAGE_CR;                         // The stopping col of the page (exclusive)
int PAGE_CC = PAGE_CL;

// Define temporary global variables
int WINDOW_SIZE_X;                   // Used for plotting
int WINDOW_SIZE_Y;
int cursorX = 10;                    // The screen coordinates
int cursorY = 10;

// The line variables
int Logic_type;                      // 0 for none, 1 for OR, 2 for AND-OR
int ABOUTFROM_SLOPE_A_AND;
int ABOUTFROM_SLOPE_B_AND;
int ABOUTFROM_SLOPE_C_AND;
int ABOUTFROM_SLOPE_A_OR;
int ABOUTFROM_SLOPE_B_OR;
int ABOUTFROM_SLOPE_C_OR;
int ABOUTFROM_SLOPE_A_SCALED;
int ABOUTFROM_SLOPE_B_SCALED;
int ABOUTFROM_SLOPE_C_SCALED;

int MAX_CHAR_LEN;    // The maximum character length in pixels

//.....
// Name: This routine will plot a scaled cimage on the screen
void clear_scale_and_plot_cimage( cimage& image)
{
  float scale, scale1, scale2;
  int NC, NY, r, c, y;
  int i, j, i1, j1, j2;

  // Compute scale to fit image on screen in x and y
  scale1 = ((float) (WINDOW_SIZE_X-20)) / ((float) image.ncol);
  scale2 = ((float) (WINDOW_SIZE_Y-20)) / ((float) image.nrow);

  if (scale2 < scale)                // Take the least of the two
    scale = scale2;
  if (scale > 1.0)                   // Only shrink the image
    scale = 1.0;

  NY = (int) ((double) image.nrow * scale);
  NX = (int) ((double) image.ncol * scale);

  xs_clr_win();                      // Clear the window
  // Draw the image
  for (r=0; r<NY; r++)
    for (c=0; c<NX; c++)
    {
      x = (int) ((double) c / scale);
      y = (int) ((double) r / scale);
      if (image.pixel(y)[x] > 0)
        xs_plot(y+10,c+10);  // border = 10;
    }

  // Draw Box Around Image
  i1 = 0;
  i2 = NY + 12;
  for (j=0; j< NX+12; j++)
  {
    if ((j<11) & 2)
      xs_plot(i1, j);
    if ((j<12) & 2)
      xs_plot(i2, j);
  }

  i1 = 0;
  i2 = NX + 12;
  for (j=0; j< NY+12; j++)
  {
    if ((j<11) & 2)
      xs_plot(j, i1);
    if ((j<12) & 2)
      xs_plot(j, i2);
  }

  xs_flush();                        // Refresh the window
}

//.....
// This routine will create a histogram from cols m to n.
// THIS WILL INSERT A 1 FOR TEXT 0 FOR BLANK!
void make_partial_histo(
    cimage& page,                    // The input image
    int m,                           // The starting and stopping col (inclusive)
    int n,                           //   (exclusive)
    int *a)                          // The output histogram (allocated previously)
{ int r,c;
```

【図200】

```
  for (i=0; i<PAGE_X1; i++)              // Zero before PAGE_X1
    a[i] = 0;
  for (i=PAGE_X1; i<PAGE_X2; i++)        // Histogram between X1 and X2
  {
    a[i] = 0;
    for (col=0; col<256; col++)
      if (page.pixel[i][col] != 0)
        {
          a[i]++;                        // Exit this search
          col = 256;
        }
  }
  for (i=PAGE_X2; i<page.size0; i++)     // Zero after PAGE_X2
    a[i] = 0;
}

//.....
// Given a range (x1,x2) and (b1, b2) will compute fraction of union/intercept
// Required: x1 <= x2 and b1 <= b2.  and the ranges overlap.
// If not overlap, will give a negative number
float overlap(
  int x1,
  int x2,                                // Range 1
  int b1,
  int b2                                 // Range 2
)
{
  int i1, i2;                            // The intercept range
  int u1, u2;                            // The union range

  if (x1 < b1) {
    i1 = b1;
    u1 = x1;
  } else {
    i1 = x1;
    u1 = b1;
  }
  if (x2 < b2) {
    i2 = x2;
    u2 = b2;
  } else {
    i2 = b2;
    u2 = x2;
  }
  if (u2 == u1)                          // Only you can prevent floating point exceptions!
    return 0.0;
  return (float) (i2-i1) / (float) (u2-u1);
}

//.....
// This routine will segment an entire page given the page
// It will call FIRST routine for each line
// Uses PAGE_X1, etc... Note: PAGE_X1 must NOT be zero, and PAGE_X2 must NOT
//     be PAGE.size0
//
void segment_page(
  char *page,                            // A characterlist (memory allocated only)
  struct charlist *clist)
{
  char line;                             // The line, one is a time
  int *startcol;                         // The start of a column (inclusive)
  int *stopcol;                          // The stop of a column (exclusive)

  int **hgram;                           // Array of histograms: [num][row]
  int **startcol;                        // Array of starting text positions
  int **stopcol;                         // Array of stopping text positions
  int *cursor;                           // The current pointer
  int *list;                             // The start of each list
  int *include;                          // Include this segment in the next line (0=no, 1=yes)
  int end_of_list;                       // Loop termination flag
  int top_div;                           // The division with the topmost text
  int range1, range2;                    // The text range in the adjacent division (row #'s
  int crange1, crange2;                  // The text range in the current division (row #'s)
  int c1, r2, c1, c2;                    // The row and column of line in page
  int minline_height;                    // The minimum line height
  int reject;                            // Line rejection flag
  int i,j,k,b,r,c;                       // Nice handy variable

  startcol = ivector(NDIVISIONS);
  stopcol = ivector(NDIVISIONS);
  hgram = imatrix(NDIVISIONS, page.size0);
  startcol = imatrix(NDIVISIONS, page.size0);  // This is overkill!
  stopcol = imatrix(NDIVISIONS, page.size0);   // This is overkill!
  cursor = ivector(NDIVISIONS);
  list = ivector(NDIVISIONS);
  include = ivector(NDIVISIONS);

  min_line_height = (int) ((float) page.dpi * PERC_OF_2_MIN_BLOCK_SIZE);

  ////////// Step 1: Define columns and find histograms //////////
  // Find range of columns
  startcol[0] = PAGE_X1;
  for (i=0; i<NDIVISIONS-1; i++)
  {
    c = PAGE_X1 + (PAGE_X2 - (i+1)) / NDIVISIONS;
    startcol[i+1] = c - OVERLAP;
    stopcol[i] = c + OVERLAP;
  }
  stopcol[NDIVISIONS-1] = PAGE_X2;
  // Find histograms
  for (i=0; i<NDIVISIONS; i++)           // For each division
  {
    make_vertical_histo(page, startcol[i], stopcol[i], hgram[i]);
  }

  ////////////////// Step 2:  Make text area lists //////////////////
  for (i=0; i<NDIVISIONS; i++)
  // Note: Zeros* are guaranteed on both ends of the histogram
  {
    size[i] = 0;
    for (j=PAGE_X1+1; j<PAGE_X2; j++)
                                         // Initialize
    {
      if ((hgram[i][j-1] == 0) && (hgram[i][j-1]) != 0)   // If starting text
        startcol[i][size[i]] = j-1;
      if ((hgram[i][j]) != 0) && (hgram[i][j+1]) == 0)    // If stopping text
        {
          stopcol[i][size[i]] = j+1;
          size[i] += 1;
        }
    }
  }

  ////////////////// Step 3: Find the lines //////////////////
  for (i=0; i<NDIVISIONS; i++)
    cursor[i] = 0;                       // Set all pointers to the top of the list
  end_of_list = 0;

  while(end_of_list == 0)
  {
```

【図201】

```
////////////// Step 3a: Find the highest text
a = 1000; // Going to be high_div
for (i=0; i<NDIVISIONS; i++)
  if (cursor[i] < size[i])          // If valid cursor
  {
    b = startext[i][cursor[i]] - stopcent[i][cursor[i]];
    if (b < a)                      // If it's better
    {
      a = b;
      top_div = i;
    }
  }

for (i=0; i<NDIVISIONS; i++)        // Initialize include[]
  include[i] = 0;                   // Don't include anything
include[top_div] = 1;               // Include the first one

r1 = startext[top_div][cursor[top_div]];   // Initialize the size of the
r2 = stopcent[top_div][cursor[top_div]];   //   change to this segment
c1 = startcol[top_div];
c2 = stopcol[top_div];

////////////// Step 3b: Loop to left and see what columns match
rangel = r1;                        // The adjacent division text range
rangel = r2;
for (i=top_div-1; i>=0; i--)
{
  if (cursor[i] < size[i])          // If we're past the end, don't do it
  {
    crangel = startext[i][cursor[i]];
    crange2 = stopcent[i][cursor[i]];
    if (overlap(rangel, rangel, crangel, crange2) > CLOSE_OVERLAP)
                                    // If overlapping
    {
      include[i] = 1;               // Include this column also
      if (crangel < r1)             // Adjust range of r1, r2, c1, c2
        r1 = crangel;
      if (crangel > r2)
        r2 = crangel;
      c1 = startcol[i];
      rangel = crangel;             // Use this for the next range
      rangel = crangel;
    }
  }
}

////////////// Step 3c: Loop to right and see what columns match
rangel = startext[top_div][cursor[top_div]];  // The adjacent division
rangel = stopcent[top_div][cursor[top_div]];  // text range.
for (i=top_div+1; i<NDIVISIONS; i++)          // Each column to the left
{
  if (cursor[i] < size[i])          // If we're past the end, don't do it
  {
    crangel = startext[i][cursor[i]];
    crange2 = stopcent[i][cursor[i]];
    if (overlap(rangel, rangel, crangel, crange2) > CLOSE_OVERLAP)
                                    // If overlapping
    {
      include[i] = 1;               // Include this column also
      if (crangel < r1)             // Adjust range of r1, r2, c1, c2
        r1 = crangel;
      if (crangel > r2)
        r2 = crangel;
      c2 = stopcol[i];
      rangel = crangel;             // Use this for the next range
      rangel = crangel;
    }
  }
}

////////////////// Step 3d: Copy the pieces into the image ////////////////////
// Optimize this.  Selectively do fill() and clipRect()
line.fill(0, r1-r1, c2-c1);         // Fill up the image with zeros
for (i=0; i<NDIVISIONS; i++)
  if (include[i] == 1)
  {
    for (textartcol[i]; c<stopcol[i]; c++)
      for (r=startext[i][cursor[i]]; r<stopcent[i][cursor[i]]; r++)
        line.pixel[r-r1][c-c1] = (int) page.pixel[r][c];
  }
line.clipspace();                   // Remove white space around the line

////////////////// Step 3e: Do we reject this image? //////////////////////
reject = 0;                         // Default is accept
if ((r2-r1) < min_line_height)
  reject = 1;
  printf(" image range is %d..%d, %d..%d\n", c1, c2, r1, r2);
//

////////////////// Step 4f: Process line ///////////////////////
if (reject == 0)
{
#ifdef VISION
  clear_pair_and_plot_image(line);
#endif
  printf("Enter 0 to run outlining character segmentation, 1 to ignore? ");
  r = auto_input_int();
  if (r == 0)
  {
    process_line(line, clist);

#ifdef SEND_OUTPUT
    send_plot_line(page, line, clist, r1, r2);
#endif
  }
}

////////////////// Skip over all the used sections
for (i=0; i<NDIVISIONS; i++)        // Up all the cursors
  if (include[i] == 1)              // Skip over the included areas
    cursor[i] += 1;

////////// Check if we've gone off the end of the list
end_of_list = 1;
for (i=0; i<NDIVISIONS; i++)
  if (cursor[i] < size[i])          // If the cursor is not at the end...
    end_of_list = 0;                // ... don't exit
}

free_vector(include);
free_vector(size);
free_vector(cursor);
free_matrix(stopcent, NDIVISIONS);
free_matrix(startext, NDIVISIONS);
free_matrix(bgrow, NDIVISIONS);
free_vector(stopcol);
free_vector(startcol);
}

//////////////////////////////////////////////////////////////////////////////
```

【図202】

```
//.....
//.....                    MAIN                    .....
//.....                                            .....
//.....                                            .....
//.....                                            .....
void xs_init()
{
  dimage page;
  struct CharList clist;
  char filename[100], filename1[100];
  int border_ignore;

#ifdef PAGE_TO_LEVEL_4
  void if_initialise();          // Initialise layer 4
  if_initialise();
#endif

  CharList_initialise(&clist, MAX_CHARS_PER_LINE, MAX_LINES_PER_HEIGHT);
  Process_line.init(0.40,2500); // Allocate the memory for the lines

  xs_win_display(WINDOW_SIZE_X, WINDOW_SIZE_Y);  // Set the window size

  printf("Input tiff filename? /d2/tif/*:");
  strcpy(filename, "/d2/tif/cosh1.tif-1");
  scanf("%s", filename1);
  strcpy(filename, "/d2/tif/");
  strcat(filename, filename1);

  auto_initialise();             // Initialise Auto-input routines

  page.dpi = 100;
// cout << "Input DPI? ";
// cin >> page.dpi;

  printf("Logic Type (0 = None, 1 = OR, 2 = AND-OR, 3 = Filled)? ");
  Logic_type = auto_input_int();

  if (Logic_type == 2)
  {
    RESOLUTION_3LINE_C_AND = RESOLUTION_3LINE_C_AND * 1;
    printf("3-line AND:\n");
    printf("  Enter 3-line constant for Rows? ");
    RESOLUTION_3LINE_C_AND = auto_input_int();
    printf("  Enter 3-line constant for Columns? ");
    RESOLUTION_3LINE_C_AND = auto_input_int();
  }

  if ((Logic_type == 1) || (Logic_type == 2))
  {
    RESOLUTION_3LINE_C_OR = RESOLUTION_3LINE_C_OR * 1;
    printf("3-line OR:\n");
    printf("  Enter 3-line constant for Rows? ");
    RESOLUTION_3LINE_C_OR = auto_input_int();
    printf("  Enter 3-line constant for Columns? ");
    RESOLUTION_3LINE_C_OR = auto_input_int();
  }

  if (Logic_type == 3)
  {
    RESOLUTION_3LINE_C_SCALED = RESOLUTION_3LINE_C_SCALED * 1;
    printf("3-line SCALING:\n");
    printf("  Enter 3-line constant for Rows? ");
    RESOLUTION_3LINE_C_SCALED = auto_input_int();
    printf("  Enter 3-line constant for Columns? ");
    RESOLUTION_3LINE_C_SCALED = auto_input_int();
  }

  MAX_COAR_LEN = 42;
  printf("Enter MAX_COAR_LEN? "); // The maximum character length in pixels
  MAX_COAR_LEN = auto_input_int();

  border_ignore = 1;
//  cout << "Input border width to ignore (in pixels)? ";
//  cin >> border_ignore;

  printf("Reading in file...");
  page.readtiff(filename);
  printf("done.\n");

#ifdef DEMO_SOFTWARE
  app_init(page);
#endif

//  border_ignore = (int) (1(inch) page.dpi * BPI_3_BORDER_IGNORE);
//  printf("Border ignore = %d\n", border_ignore);

  PAGE_A1 = border_ignore;
  PAGE_A1 = page.size - border_ignore;
  PAGE_A0 = PAGE_A1 - PAGE_A1;
  PAGE_C1 = border_ignore;
  PAGE_C1 = page.size - border_ignore;
  PAGE_C0 = PAGE_C1 - PAGE_C1;

  // ------------ THIS IS JUST FOR TESTING ------------------------
  int i;
  xs_flush();
  printf("Enter 0 to continue? ");
  i = auto_input_int();

  // clear_scale_and_plot_clusters(page);

  printf("Pausing because page is displayed. Input 0 to segment page? ");
  border_ignore = auto_input_int();

  while(1) {
    segment_Page(page, &clist);
    xs_flush();

    printf("Program finished. Enter 0 to run again? ");
    int ttemp;
    ttemp = auto_input_int();
  }

  CharList_terminate(&clist);
  auto_terminate();
}
```

【図２０４】

```
//
// Filename: seiagprint.c
// This contains routines to demonstrate the final character segmentation
//
////////////////////////////////////////////////////////////////////////
//
#include "wtoinput.h"
#include "xm.h"
#include "image.h"
#include "chatlist.h"

#define SCALE 2                 // Set this to adjust page scale
#define LEFT_EDGE 10.           // The left edge to plot the characters

extern int WINDOW_SIZE_X;
extern int WINDOW_SIZE_Y;

// Define global static variables (the characters)
int segm_curx;                  // The cursor positions of the
int segm_cury;                  // segmented characters
int segm_bot_row_printed;       // The bottom most row printed in the top 1/2

// Define Colors
#define LINES_R 255
#define LINES_G 0
#define LINES_B 0
#define TEXT_R 0
#define TEXT_G 0
#define TEXT_B 0
#define BACK_R 255
#define BACK_G 0
#define BACK_B 0

////////////////////////////////////////////////////////////////////////
// This routine will plot a portion of scaled image on the screen.
int segm_plot_page(             // Returns the max row printed
  image page,                   // The page to plot
  int start_row)                // The starting row to start printing from
{
  int right_side;               // The border just to the right of the image
  int c,x,y;

  x = y = 10;                   //

  if (start_row < 0)            // Force it to be in range
    start_row = 0;

  right_side = 10 + page.size.x/SCALE;

  xm_set_foreground(LINES_R, LINES_G, LINES_B);
  for (c=0; c<WINDOW_SIZE_X; c++)
    xm_plot(c, WINDOW_SIZE_Y/2);        // Plot middle line

  if (start_row < 1)
    for (c=0; c<right_side; c++)        // Plot top dotted line
      if ((c+1)%2)
        xm_plot(c,9);

  for (restart_row; rspage.size.y<=SCALE)
  {

    xm_set_foreground(LINES_R, LINES_G, LINES_B);
    x = 10;
    if ((y+1)%2)                        // Plot left dotted line
      xm_plot(9,y);
    if ((x+right_side)%2)               // Plot right dotted line
      xm_plot(right_side,y);

    xm_set_foreground(TEXT_R, TEXT_G, TEXT_B);
    for (c=0; c<page.size.x; c++)
    {
      if (page.pixel[y][c] != 0)
        xm_plot(x,y);
      x++;
    }
    y++;
    if (y > WINDOW_SIZE_Y/2 - 10)       // Exit if at middle of page
      return y;
  }

  xm_set_foreground(LINES_R, LINES_G, LINES_B);
  for (c=0; c<right_side; c++)          // Print bottom line
    if ((c+y)%2)
      xm_plot(c,y);

  return y;
}

////////////////////////////////////////////////////////////////////////
// The main routine
////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////
// This routine will initialize the screen for plotting lines on it
void segm_init(clear page)
{
  segm_bot_row_printed = -1;    // Make sure it reprints the top part
  xm_clear();
  segm_bot_row_printed = segm_plot_page(page, 0);
  segm_curx = LEFT_EDGE;
  segm_cury = WINDOW_SIZE_Y/2 + 20;
  xm_flush();
}

////////////////////////////////////////////////////////////////////////
// This routine will print another segmented line on the screen.
// updating the screen if necessary
void segm_plot_line(
  image page,                   // The plot
  int line_no,                  // The line we're segmenting
  struct chatlist *clist,       // The character list to print
  int line_top,                 // The top of the line being printed
  int line_bot)                 // The bottom of the line being printed
{
  struct Character *chr;        // The character we're on
  int max_c_height;             // The highest character height
  int new_curx;                 // The new cursor location (or next cha
  int last_right_edge;          // The right edge of the prev character
  int x,y,r,c;
  int y1,y2;

  int color;                    // Test stuff

  color = 0;
  // Find maximum height of the characters in this line and put it in
  // it max_c_height
  max_c_height = 0;
  chr = clist->first;
  while (chr != 0)
  {
```

【図205】

```
    if (ch->box - chr->tool > MAX_C_WIDGET)
      max_height = ch->box - chr->sep
    ch = ch->next;
  }

#ifdef VISIBLE
  sesp_box_row_printed = -1;    // Make sure it erases if visible
#endif

  // Check if we need to replot the entire screen
  if ((line_box = sesp_box_row_printed) ||
      (sesp_curs - line.size/2 > WINDOW_SIZE_Y-10))
  {
    printf("Going to scroll... Enter 0 to continue? ");
    scanf("%d", &x);
    xs_clr_win();                 // Clear the window
    sesp_box_row_printed = sesp_plot_page(page, line_top-20);
    sesp_curs = LEFT_EDGE;        // Set the new cursor
    sesp_curr = WINDOW_SIZE_Y/2 - 20;
  }

  // Print the character list
  chr = clist->first;

  while (chr != 0)
  {
    // Check if need to scroll one line
    if (sesp_curs + chr->right + chr->left + 20 > WINDOW_SIZE_X)
    {
      sesp_curs = sesp_curs - height/SCALE + 10;   // Move down
      sesp_curs = LEFT_EDGE + 50;                  // Indent
    }
    y1=19999;
    x = sesp_curs; y = sesp_curr;   // Initialize cursor
    xs_sel_foreground(TEXT_S, TEXT_B);
    for (r=0; r<line.size-1; r++SCALE)
    {
      if ((r >= chr->top) && (r <= chr->bot))
      {
        if (y < y1)                 // Find y1, 1st row printed
          y1 = y;
        y2 = y;                     // Set y2 = last row printed
        for (c=chr->left; c < chr->right; c+=SCALE)
        {
          if ((c >= chr->curve[r]) && (c < chr->curve[r+1]))
            if (line.pixel[r][c] & 1)
              xs_plot(x,y);

          x++;
        }
        new_curs = x;
        x = sesp_curs;
      }
      y++;
    }
    // Draw Box
    xs_set_foreground(BOX_S, BOX_G, BOX_B);
    for (x=sesp_curs-2; x<new_curs+2; x++)
    {
      xs_plot(x, y1-2);
      xs_plot(x, y2+2);
    }
    for (y=y1-2; y<=y2+2; y++)
    {
      xs_plot(sesp_curs-2, y);
      xs_plot(new_curs+2, y);
    }

    chr = chr->next;
  }

  ////////////// Get next character and move cursor over

  last_right_edge = chr->right;

  // Get next cursor
  chr = chr->next;

  // Update the cursor (slide to right)
  if (chr)
    if (chr->left - last_right_edge-2) < 3)
      sesp_curs = new_curs + 3;
    else
      sesp_curs = new_curs - chr->left - last_right_edge + 2;

  xs_flush();

  sesp_curr -= max_c_height/SCALE + 10; // Reposition the cursor for next line
  sesp_curs = LEFT_EDGE;

#ifdef VISIBLE
  printf("Enter 0 to continue? (not auto input)) ");
  scanf("%d", &x);
#endif
}
```

【図２０６】

```
/* This program will test the charlist */
#include"charlist.h"

void print_charlist(clist)
  struct Charlist *clist;
{
  struct Character *ch;

  printf("Charlist:\n ");
  ch = clist->first;

  while(ch != 0) {
    printf("  Loc: %d: Chars (+,-) = (", ch->loc/8);
    if (ch->prev == 0)
      printf("***");
    else
      printf("%d",ch->prev->top);
    printf(",%d,",ch->top);
    if (ch->next == 0)
      printf("***");
    else
      printf("%d",ch->next->top);
    printf(")\n");

    ch = ch->next;
  }
  printf("End of charlist\n\n");
}


main()
{
  struct Character *ch;
  struct Charlist clist;
  int d[10];                    /* Dummy vector for height */
  int i,p;

  Charlist_initialize(&clist, 10, 10);

  for (i=0; i<3; i++)
  {
    printf("Position of new char %d:) ", i);
    scanf("%d", &p);
    ch = &(clist.chr[clist.size]);
    ch->top = 1;
    ch->bot = i-1;
    ch->curve1[i] = ch->curve2[i] = p;
    Charlist_place_new_char(&clist);
    print_charlist(&clist);
  }

  while(1)
  {
    printf("Replace which char) ");
    scanf("%d", &i);

    ch = clist.first;
    while(ch != 0) {
      if (ch->top == i)
        break;
      ch = ch->next;
    }
    if (ch == 0) {
      printf("Character not found.\n");
    } else {
      printf("New position? ");
      scanf("%d", &p);
      ch->curve1[i] = ch->curve2[i] = p;
      Charlist_replace_char(&clist, ch);
      print_charlist(&clist);
    }
  }
}
```

【図２０９】

```
/* This file prints a character to be recognised on the screen by */
/* calling the routine thiel_recognize() */

void toshi_recognise(size_r, size_c, image)
     int size_r;                    /* The number of rows */
     int size_c;                    /* The number of columns */
     char **image;              /* The image: Bit 1 is the image */
{
  int r,c;

  printf("\nThe character is:\n");
  for (r=0; r<size_r; r++)
  {
    for (c=0; c<size_c; c++)
      if (image[r][c] & 1)
        printf("*");            /* On */
      else
        printf(".");            /* Off */
    printf("\n");
  }
  printf("****** End of character *******\n\n");
}
```

【図207】

```
//
// Filename: test_mult.C
// This file is used to test and develop multilevel2.C
////////////////////////////////////////////////////////////////////
//
#include <stdio.h>
#include "cimages.h"
#include "mymalloc.h"
#include "xs.h"
#include "charlist.h"

extern void l4_initialise();
void level4(cimage& limg, struct Charlist *clist);

void xs_init()
{
  struct Charlist clist;
  struct Character *ch;
  cimage image;
  int i,j;

  l4_initialise();

  Charlist_initialise(&clist, 10, 300); // 10 chars, 100-max height
  ch = &(clist.chr[clist.size]); // Set new_char to the last character

  ////////////////////////////////////////////////////////////////////
  // Load the image
  //
  image.fill(1,10,10);
  image.pixel[6][6] = 0;
  image.pixel[7][6] = 0;
  image.pixel[8][6] = 0;
  image.pixel[6][7] = 0;
  image.pixel[7][7] = 0;
  image.pixel[8][7] = 0;
  image.pixel[6][8] = 0;
  image.pixel[7][8] = 0;
  image.pixel[8][8] = 0;
  image.pixel[6][9] = 0;
  image.pixel[7][9] = 0;
  image.pixel[8][9] = 0;
  image.pixel[2][10] = 0;
  image.pixel[3][10] = 0;
  image.pixel[4][10] = 0;
  image.pixel[5][10] = 0;
  image.pixel[6][10] = 0;
  image.pixel[7][10] = 0;
  image.pixel[8][10] = 0;
  image.pixel[2][11] = 0;
  image.pixel[3][11] = 0;
  image.pixel[4][11] = 0;
  image.pixel[5][11] = 0;
  image.pixel[6][11] = 0;
  image.pixel[7][11] = 0;
  image.pixel[8][11] = 0;
  image.pixel[0][12] = 0;
  image.pixel[1][12] = 0;
  image.pixel[2][12] = 0;
  image.pixel[3][12] = 0;
  image.pixel[0][13] = 0;
  image.pixel[1][13] = 0;
  image.pixel[2][13] = 0;
  image.pixel[0][14] = 0;
  for (i=1; i<9; i++)
    image.pixel[i][20] = 0;
  for (i=0; i<10; i++)
    image.pixel[i][21] = 0;
  for (i=1; i<9; i++)
    image.pixel[i][22] = 0;
  for (i=27; i<30; i++)
    for (j=3; j<8; j++)
      image.pixel[j][i] = 0;
  for (i=25; i<28; i++)
    for (j=1; j<2; j++)
      image.pixel[j][i] = 0;
  for (i=29; i<31; i++)
    for (j=7; j<9; j++)
      image.pixel[j][i] = 0;

  // Load the character
  ch->top = 0;
  ch->bot = 10;
  ch->left = 0;
  ch->right = 15;
  for (i=ch->top; i<ch->bot; i++) {
    ch->curve1[i] = ch->left;
    ch->curve2[i] = ch->right;
  }

  Charlist_place_new_char(&clist);

  level4(image, &clist);

  printf("**** End of Program ****\n");
}
```

【図214】

```
/* If your routines are in C++ then CPP must be defined in make file */
#ifdef CPP
  extern "C" void xs_set_foreground(int r, int g, int b);
  extern "C" void xs_set_background(int r, int g, int b);
  extern "C" void xs_init();
  extern "C" void xs_flush();
  extern "C" void xs_plotxy(int x,int y);
  extern "C" void xs_unplotxy(int x,int y);
  extern "C" void xs_redisplay();
  extern "C" void xs_cli_bitmap(int x,int y,int width,int height,char *buf);
  extern "C" void xs_clr_win();
  extern "C" void xs_draw_box(int x,int y,int width,int height);
  extern "C" void xs_string(int x,int y,char *str);
  extern "C" void xs_setcolor(int color);
  extern "C" void xs_win_dim(int *width,int *height);
#endif
```

【図２１０】

```
/*
FILE: util.c

COMMENTS:
        This file contains general utility routines.

        ut_reverse(width,height,buf);
        ut_getbits(x,p,n);
        ut_ascii_bitmap(width,height,buf);

    Usage:

        Just call the routines.

*/

/* ---------------------------Includes------------------------------- */
#include stdio.h>
#include malloc.h>

/* ---------------------------Global--------------------------------- */

/* ---------------------------Extern--------------------------------- */

/*
Routine: ut_reverse(width,height,buf)

Description: Reverse the bits for display format

Return: Nothing

Author: Rick Lee
*/
ut_reverse(width,height,buf)
int width;
int height;
char *buf;
{
    unsigned char val,new_val;
    int x,y,j;

    for(x = 0; x < (width>>3)*height; x++)
    {
        j = 0;
        val = buf[x];
        new_val = 0;
        for(y = 7; y >= 0 ; y--)
        {
            if(ut_getbits(val,x++,1))
                new_val = 0x01 << y | new_val;
        }
        buf[x] = new_val;
    }
}

/*
Routine: ut_getbits(x,p,n)

Description: Check if the bit is turned on or not

Return: Nothing

Author: Rick Lee
*/
ut_getbits(x,p,n)
unsigned char x;    /* The character to slice bit(s) from */
int p;              /* The beginning of the bit number */
int n;              /* Number of bits to get */
{
    return(x >> (p+1-n)) & ~(~0 << n);
}

/*
Routine: ut_ascii_bitmap(width,height,buf)

Description: Display the bitmap using ascii characters

Return: Nothing

Author: Rick Lee
*/
ut_ascii_bitmap(width,height,buf)
int width;
int height;
char *buf;
{
    short bit_code = 0x8000;
    short bit_result,i=0;
    short x,y,z,bits;

    for(x = 0; x < height; x++)
    {
        for(y = 0; y < width; y++)
        {
            bits = buf[i++];
            for(z = 0; z < 8; z++)
            {
                bit_result = bits & bit_code;
                if(bit_result != 0)
                    printf("1");
                else
                    printf("0");
                bit_code = bit_code >> 1;
            }
            bit_code = 0x8000;
        }
        printf("\n");
    }
}
```

[図211]

```
/*
FILE: xs.C

COMMENTS:
      This file contains routines for displaying 'things'.

      xs_flush();
      xs_plotxy(x,y);
      xs_unplotxy(x,y);
      xs_redisplay();
      xs_dis_Bitmap(x,y,width,height,buf);
      xs_clr_win();
      xs_draw_box(x,y,width,height);
      xs_string(x,y,str);
      xs_setcolor(color);
      xs_win_dim(width,height);

  Usage:

      Just call these routines once the hook has been made with the
      display routine.
*/

/* ----------------------Includes---------------------- */
#include <stdio.h>
#include <stdlib.h>
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <Xm/Xm.h>
#include <Xm/DrawingA.h>
#include <Xm/PushB.h>
#include <Xm/Form.h>
#include <X11/Util.h>

/* ----------------------Global Var---------------------- */


/* ----------------------Externs---------------------- */
extern Widget parent,           /* The root widget */
      canvas;                   /* The display widget */
extern GC line_gc;              /* The graphic context (colors) */
extern XGCValues gcv;


/************************** ROUTINES ************************/


/* Internal Utility to create/return the pixel of an RGB value */
int xs_get_color(red, green, blue) /* Returns a colormap entry */
  int red, green, blue;         /* From 0 to 255! */
{
  XColor color;
  Colormap cmap;
  Display *dpy = XtDisplay(canvas);

  cmap = DefaultColormap(dpy, DefaultScreen(dpy));

  color.red = red*256;
  color.green = green*256;
  color.blue = blue*256;

  if (XAllocColor(dpy, cmap, &color))
    return (color.pixel);

  printf("Warning: Couldn't allocate requested color\n");
  return BlackPixel(dpy, DefaultScreen(dpy));
}


/* Routine to set printing color */
void xs_set_foreground(red, green, blue)
  int red, green, blue;         /* From 0 to 255! */
{
  XSetForeground(XtDisplay(canvas),
                 line_gc,
                 xs_get_color(red, green, blue));
}


/* Routine to clear window and set background color */
void xs_set_background(red, green, blue)
  int red, green, blue;         /* From 0 to 255! */
{
  XSetBackground(XtDisplay(canvas),
                 line_gc,
                 xs_get_color(red, green, blue));
  xs_clr_win();                 /* Erase it! */
  xs_flush();
}


xs_flush()
{
   /* Flush the queue to X-server */
   XFlush(XtDisplay(canvas));
}


/*
Routine: xs_plotxy(x,y)

Description: Draw a point on the screen

Return: Nothing

Author: Rick Lee
*/
xs_plotxy(x,y)
int x;
int y;
{
   XDrawPoint(XtDisplay(canvas),XtWindow(canvas),line_gc,x,y);
}

/*
Routine: xs_unplotxy(x,y)

Description: UNDraw a point on the screen

Return: Nothing

Author: Rick Lee
*/
xs_unplotxy(x,y)
int x;
int y;
{
   XSetForeground(XtDisplay(canvas),line_gc,gcv.background);
   XDrawPoint(XtDisplay(canvas),XtWindow(canvas),line_gc,x,y);
   XSetForeground(XtDisplay(canvas),line_gc,gcv.foreground);
}
```

【図212】

```
/*
Routine: xm_redisplay()

Description: Redisplay the 'stuff' that was on the window

Returns: Nothing

Author: Rick Lee
*/
xm_redisplay()
{
}


/*
Routine: xm_dis_bitmap(x,y,width,height,buf)

Description: Draw the given bitmap

Returns: Nothing

Author: Rick Lee
*/
xm_dis_bitmap(x,y,width,height,buf)
{
    Pixmap bitmap,pix;   /* The bitmap for display */

    /* Reverse the bit ordering */
    uc_reverse(width,height,buf);

    /* First create bitmap */
    bitmap = XCreateBitmapFromData(XtDisplay(canvas),
                    RootWindowOfScreen(XtScreen(canvas)),
                                   buf,width,height);

    /* Create the pixmap area */
    pix = XCreatePixmap(XtDisplay(canvas),RootWindowOfScreen(XtScreen(canvas)),width,
height,
                        DefaultDepthOfScreen(XtScreen(canvas)));

    /* Now copy the data to bitmap */
    XCopyPlane(XtDisplay(canvas),bitmap,pix,line_gc,0,0,
               width,height,0,0,1);

    /* Now copy the pixmap to drawing area */
    XCopyArea(XtDisplay(canvas),pix,XtWindow(canvas),line_gc,0,0,
              width,height,x,y);

    /* Now free the bitmap */
    XFreePixmap(XtDisplay(canvas),bitmap);

    /*  Now free the bitmap */
    XFreePixmap(XtDisplay(canvas),pix);

    /* Now flush the queue to X-server */
    XFlush(XtDisplay(canvas));
}


/*
Routine: xm_clr_win()

Description: Clear the screen

Return: Nothing

Author: Rick Lee
*/
xm_clr_win()
{
    XClearArea(XtDisplay(canvas),XtWindow(canvas),0,0,0,0,TRUE);
}


/*
Routine: xm_draw_box(x,y,width,height)

Description: Draw a box on screen

Returns: Nothing

Author: Rick Lee
*/
xm_draw_box(x,y,width,height)
int x;
int y;
int width
int height.
{
    XDrawRectangle(XtDisplay(canvas),XtWindow(canvas),line_gc,
                                x,y,width,height);
}


/*
Routine: xm_string
Description: Draw a string on the screen

Returns: Nothing

Author: Rick Lee
*/
xm_string(x,y,str)
int x;
int y;
char *str;
{
    XDrawString(XtDisplay(canvas),XtWindow(canvas),line_gc,
                x,y,str,strlen(str));
}


/*
Routine: xm_setcolor

Description: Set the current color.

Return: Nothing

Author: Rick Lee
*/
xm_setcolor(color)
int color;
{
    gcv.foreground = color;
    XSetForeground(XtDisplay(canvas),line_gc,color);
}
```

【図213】

```
/*
 Routine: xs_win_dim

 Description: Get the size of the window

 Return: Nothing

 Author: Rick Lee
 */
xs_win_dim(width,height)
int *width;
int *height;
{
   Arg args[2];
   Dimension win_width,win_height;

   XtSetArg(args[0],XtNwidth,&win_width);
   XtSetArg(args[1],XtNheight,&win_height);
   XtGetValues(canvas,args,2);

   *width = win_width;
   *height = win_height;
}

/*
xs_set_cmap(w)
Widget w;
{
   int ncolors;
   XColor colors[256];
   Colormap my_cmap;
   int count = 1;


   Display *d;
   int scr;
   Colormap def_colormap;
   XColor Color;
   int x;
   int the_color = 3000;

   d = XtDisplay(w);
   scr = DefaultScreen(d);
   ncolors = DisplayCells(d,scr);
   def_colormap = DefaultColormap(d,scr);
   for(x = 0; x < ncolors; x++)
   {
      colors[x].pixel = x;
      colors[x].flags = DoRed | DoGreen | DoBlue;

      colors[x].red = the_color;
      colors[x].green = the_color;
      colors[x].blue = the_color;
      the_color += 500;
   }
   XQueryColors(d,def_colormap,colors,ncolors);

   my_cmap = XCreateColormap(d,DefaultRootWindow(d),DefaultVisual(d,scr),
                             AllocAll);
   XStoreColors(d,my_cmap,colors,ncolors);
}
*/
```

【図228】

```
   c1 = get2bytes(f, fp);
   c2 = get2bytes(f, fp);
   return c1 + c2*65536;
}


////////////////////////////////////////////////////////////////////////
// Plot image using Flash Graphics at (xloc, yloc) with color
void cimage::plot(int x, int y)
{
  int px,py;            // Pixel coords on the screen (upper left)
  char color;

  for (py=0; py<sizeR; py++)
    for (px=0; px<sizeC; px++)
      {
        color = pixel[py][px];
        if (color & 1)
          xs_plotxy(x+px, y+py); // Draw it
      }
};


////////////////////////////////////////////////////////////////////////
// Plot image with a box around it (with color 1!)
void cimage::plotbox(int x, int y)
{
  int px,py;

  for (px=0; px<sizeC+2; px++) {        // Draw horizontal lines
    if ((x+px+y)%2 == 0)
      xs_plotxy(x-px, y);
    if ((x+px+y+sizeR+1)%2 == 0)
      xs_plotxy(x+px, y+sizeR+1);
  }
  for (py=1; py<sizeR+1; py++) {
    if ((x+y+py)%2 == 0)
      xs_plotxy(x, y+py);
    if ((x+sizeC+1+y+py)%2 == 0)
      xs_plotxy(x+sizeC+1, y+py);
  }
  plot(x+1, y+1);       // Plot the image
}
```

【図240】

```
/* This file prints a character to be recognized on the screen by */
/* calling the routine chieu_recognize() */

char toshi_recognize(size_r, size_c, image)
     int size_r;                    /* The number of rows */
     int size_c;                    /* The number of columns */
     char **image;              /* The image: Bit 1 is the image */
{
  int r,c;
  char input_string[100];

  printf("\nThe character is:\n");
  for (r=0; r<size_r; r++)
  {
    for (c=0; c<size_c; c++)
      if (image[r][c] & 1)
        printf("*");            /* On */
      else
        printf(".");            /* Off */
    printf("\n");
  }
  printf("****** End of character *******\n\n");
  printf("Enter Character? ");
  scanf("%s", input_string);
  return input_string[strlen(input_string)-1];
}
```

【図215】

```
/*
File: xt.C

COMMENTS:
    This file contains interface to X-display routines.

        main();
        xt_redisplay(w,data,call_data);

    Usage:
        Your initialize routine must be xt_init().

*/

#define DEFAULT_WINDOW_SIZE_HEIGHT 800
#define DEFAULT_WINDOW_SIZE_WIDTH 1330

/* ------------------------------------ */
#include <stdio.h>
#include <stdlib.h>
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <Xm/Xm.h>
#include <Xm/DrawnB.h>
#include <Xm/Form.h>
#include <X11/Xutil.h>

/* ------------------------------------ */
Widget parent,      /* The root widget */
       canvas;      /* The widget to display */
int start_up = 1;   /* The flag to call init routine */
GC line_gc;         /* Points to GC */
XGCValues gcv;      /* The GC values */

/* ------------------------------------ */
Externs

/*
Routine: xt_redisplay(w,data,call_data)

Description: Expose callback

Return: Nothing

Author: Rick Lee
*/
void xt_redisplay(w,data,call_data)
Widget w;           /* The widget associated with display */
char *data;         /* The data */
XmDrawingAreaCallbackStruct *call_data;  /* Not used */
{
    if(start_up)
    {
        start_up = 0;
        xt_init();
    }
}

main(argc,argv)
int argc;
char *argv[];
{
    Arg args[10];
    int n,i;
    int no_set_win_size;        /* 1 to not set window size */
    int no_set_win_bg;          /* 1 to not set window background */

#ifdef CPP
    /* Call the routine to setup C++ interface, take out if no C++ interface */
    _main();
#endif

    /* Checks for geometry and background options in command line */
    no_set_win_size = 0;
    no_set_win_bg = 0;
    for (i=0; i<argc; i++) {
        if (strcmp(argv[i],"-geometry") == 0)
            no_set_win_size = 1;
        if (strcmp(argv[i],"-bg") == 0)
            no_set_win_bg = 1;
        if (strcmp(argv[i],"-width") == 0)
            no_set_win_size = 1;
        if (strcmp(argv[i],"-height") == 0)
            no_set_win_size = 1;
        if (strcmp(argv[i],"-background") == 0)
            no_set_win_bg = 1;
        if (strcmp(argv[i],"-bg") == 0)
            no_set_win_bg = 1;
    }

    /* Init the Xt function */
    parent = XtInitialize(argv[0],"Pixmap", NULL, 0, &argc,argv);

/*  XtGetApplicationResources(parent, 0, NULL, 0, args, n); */
/*
    XSetBackground(XtDisplay(parent), line_gc,
                   WhitePixel(XtDisplay(parent),
                              DefaultScreen(XtDisplay(parent))));
*/
    /* Create drawing area widget */
    canvas = XtCreateManagedWidget("canvas",xmDrawingAreaWidgetClass,
                    parent,NULL,0);

    /* Set the GC for display routines */
    line_gc = XCreateGC(XtDisplay(canvas),
                        DefaultRootWindow(XtDisplay(canvas)),
                        0, &gcv);

    /* Now add callbacks and event handler */
    XtAddCallback(canvas,XmNexposeCallback,xt_redisplay,NULL);

    /* Set the default screen size */
    n=0;
    if (no_set_win_size == 0) {
        XtSetArg(args[n], XtNwidth, DEFAULT_WINDOW_SIZE_WIDTH); n++;
        XtSetArg(args[n], XtNheight, DEFAULT_WINDOW_SIZE_HEIGHT); n++;
    }
    if (no_set_win_bg == 0)
        XtSetArg(args[n], XtNbackground, WhitePixel(XtDisplay(parent), DefaultScreen(Xt
    XtSetValues(parent, args, n);

    /* Display the widget and enter event loop */
    XtRealizeWidget(parent);

    XtMainLoop();
}
```

【図216】

```
//
// Filename: zerovpp.C
// This routine contains the lines to find blocks that have non-zero vpp
// (Level 1 segmentation)
//
////////////////////////////////////////////////////////////////////////
//
#include"charlist.h"
#include"cimages.h"

////////////////////////////////////////////////////////////////////////
// Scan a line quickly (every 1/4 pixel) looking for an 'on' pixel
int line_scan_quick(    // Returns 1 for pixel found, 0 otherwise
  cimage4 line,         // The line to process
  int c)                // The column to scan
{
  int r;

  for (r=0; r<line.sizeR; r+=4)
    if ((line.pixel[r][c] & 1)
      return 1;
  return 0;
}


////////////////////////////////////////////////////////////////////////
// Scan a line completely looking for an 'on' pixel
int line_scan_complete( // Returns 1 for pixel found, 0 for otherwise
  cimage4 line,         // The line
  int c)                // The column to scan
{
  int r;

  for (r=0; r<line.sizeR; r++)
    if ((line.pixel[r][c] & 1)
      return 1;
  return 0;
}


////////////////////////////////////////////////////////////////////////
// The main routine.  Call this to make the list
void extract_zero_vpp_list(
  cimage4 line,         // The image of the line to segment
  struct NonZerovppList *list)  // A pointer to the list
{
  int c;
  int left, right;

  for (c=0; c<line.sizeC; c+=4)  // Scan the image
  {
    if (line_scan_quick(line, c))  // If found something
    {
      left = right = c;         // Look to the left for the start
      while (--c >= 0)
        if (line_scan_complete(line, c) == 0)
          break;
      left = c+1;
      c = right;                // Keep looking to the right
      while (++c < line.sizeC)
        if (line_scan_complete(line, c) == 0)
          break;
      right = c;
      zerovpp_add(list, left, right);   // Add in the new break
    }
  }
```

【図245】

```
/* If your routines are in C++ then CPP must be defined in make file */
#ifdef CPP
   extern "C" void xs_init();
   extern "C" void xs_flush();
   extern "C" void xs_plotxy(int x,int y);
   extern "C" void xs_unplotxy(int x,int y);
   extern "C" void xs_redisplay();
   extern "C" void xs_dis_bitmap(int x,int y,int width,int height,char *buf);
   extern "C" void xs_clr_win();
   extern "C" void xs_draw_box(int x,int y,int width,int height);
   extern "C" void xs_string(int x,int y,char *str);
   extern "C" void xs_setcolor(int color);
   extern "C" void xs_win_dim(int *width,int *height);
#endif
```

【図218】

```
CC=     cc
CPP=    CC
COMPILE_PROFILE = #-p
COMPILE_STYLE = -g # -O
COMPILE_VISIBLE = -DVISIBLE
COMPILE_MAKE_DBASE = #-DMAKE_DATABASE
LDLIBS= -lm
LDLIBSX = -lXm -lXt -lX11 -ll -lmalloc -lPW
CPPFLAGS= -DCPP $(COMPILE_STYLE) $(COMPILE_VISIBLE) $(COMPILE_PROFILE) $(COMPILE_MAKE
_DBASE)
CFLAGS= $(COMPILE_STYLE) $(COMPILE_VISIBLE) $(COMPILE_PROFILE) $(COMPILE_MAKE_DBASE)
OBJECTS_courcut = psegment.o lineprocess.o zerovpp.o charlist.o cimages.d mymalloc.o
xs.o xt.o util.o autoinput.o courcut.o thisu_file.o

all: courcut

courcut: $(OBJECTS_courcut)
        $(CPP) -o courcut $(OBJECTS_courcut) $(LDLIBS) $(LDLIBSX)

autoinput.o: autoinput.c
        $(CC) -c $(CFLAGS) autoinput.c

cimages.o: cimages.C
        $(CPP) -c $(CPPFLAGS) cimages.C

courcut.o: courcut.C
        $(CPP) -o $(CPPFLAGS) courcut.C

lineprocess.o: lineprocess.C
        $(CPP) -c $(CPPFLAGS) lineprocess.C

mymalloc.o: mymalloc.c
        $(CC) -c $(CFLAGS) mymalloc.c

psegment.o: psegment.C
        $(CPP) -c $(CPPFLAGS) psegment.C

scaleprint.o: scaleprint.C
        $(CPP) -c $(CPPFLAGS) scaleprint.C

thisu_file.o: thisu_file.c
        $(CC) -c $(CFLAGS) thisu_file.c

util.o: util.c
        $(CC) -c $(CFLAGS) util.c

xs.o: xs.c
        $(CC) -c $(CFLAGS) xs.c

xt.o: xt.c
        $(CC) -c -DCPP $(CFLAGS) xt.c

zerovpp.o: zerovpp.C
        $(CPP) -c $(CPPFLAGS) zerovpp.C

charlist.o: charlist.h mymalloc.h
combine.o: charlist.h cimages.h
cimages.o: cimages.h xs.h
level4.o: mymalloc.h
lineprocess: charlist.h cimages.h
multilevel.o: charlist.h xs.h mymalloc.h
multilevel2.o: charlist.h cimages.h xs.h mymalloc.h
outline.o: charlist.h cimages.h
psegment.o: cimages.h xs.h mymalloc.h
level3ml.o: charlist.h cimages.h
zerovpp.o: charlist.h cimages.h

rm_visible:
        rm lineprocess.o psegment.o
```

【図219】

```
/*
 * Filename: autoinput.h
 * This file contains the code to input from a file/ess
 *.....
 */

#include <stdio.h>
#include "autoinput.h"

/* Global variables */

int auto_eof_detected;          /* 1: 0 if eof has been detected */
FILE *auto_fp;                  /* The file pointer */
int save_to_file;               /* Saving the input to a file? */
int save_count;                 /* Number of inputs before newline */
char filename[100];             /* The filename */

/*.....*/
void auto_initialize()
{
  int not_exit = 1;

  auto_eof_detected = 0;

  while(not_exit)
  {
    printf("Input filename for auto-input? ");
    scanf("%s", filename);

    auto_fp = fopen(filename, "r");

    if (auto_fp == NULL)        /* Read open failed */
    {
      auto_fp = fopen(filename, "w");
      if (auto_fp == NULL)      /* Write open failed */
      {
        printf("Cannot open file '%s'\n", filename);
      } else {                  /* Opened new file */
        printf("'%s' opened as a new file.\n", filename);
        printf("Enter 1 at the next prompt.\n");
        save_to_file = 1;
        auto_eof_detected = 1;
        not_exit = 0;
      }
    } else {                    /* Read open succeeded */
      printf("Opening existing file %s\n",filename);
      not_exit = 0;
    }
  }

  printf("Do you wish to save the auto-input into this file? (1 = yes)? ");
  scanf("%d", &save_to_file);
  auto_input_int();
  printf("Maintain change the file '1' to a '0' to turn off appending to the file\n");
  save_count = 0;
}


int auto_input_int()
{
  int i;

  if (auto_eof_detected)
  {
    printf("\n[Manual Input (-9 = newline, -9999 = exit)] ? ");
    scanf("%d", &i);
    if (i == -9999)
    {
      fclose(auto_fp);
      exit(0);
    }
    if (save_to_file == 1)
    {
      if (i == -9)              /* Print new lines */
      {
        fprintf(auto_fp, "\n"); /* Print the new lines */
        fprintf(auto_fp, "\n");
        fflush(auto_fp);
        save_count = 0;
        return auto_input_int(); /* And input again */
      }
      fprintf(auto_fp, "%d ", i);
      fflush(auto_fp);
      if (++save_count > 10)
      {
        save_count = 0;
        fprintf(auto_fp, "\n");
      }
    }
    return i;
  }

  if (feof(auto_fp, "%d", &i) == EOF)
  {
    printf("\n**** END OF FILE DETECTED ****\n");
    if (save_to_file == 1)
    {
      fclose(auto_fp);
      auto_fp = fopen(filename, "a");
      if (auto_fp == NULL)
      {
        printf("****** WARNING: CANNOT APPEND TO FILE: %s ******\n", filename);
        save_to_file = 0;
      }
      fprintf(auto_fp, "\n\n");
    }
    auto_eof_detected = 1;
    return auto_input_int();
  }
  printf("%d\n",i);
  return i;
}

void auto_terminate()
{
  fclose(auto_fp);
}
```

【図221】

```
/*
 * Filename: charlist.c
 * This file contains charlist and fontcurvyplist utility routines.
 *
 *
 */

/*-------------------------------------------------------------------*/
/*                  Routines to handle Character lists               */
/*-------------------------------------------------------------------*/
/* By Christopher Sherrick */

#include"charlist.h"
#include"sysalloc.h"

/*-------------------------------------------------------------------*/
/* Initialize and allocate a Characterlist (set up curve pointers) */
/* Call this routine once only! */
void Charlist_Initialize(clist, clist_max, vector_height)
    struct Charlist *clist;     /* The Characterlist */
    int clist_max;              /* The max number of characters */
    int vector_height;          /* Height of the line (to allocate curves) */
{
  int i;                        /* A looping var */

  /* Allocate memory for characters */
  clist->ch = (struct Character *) malloc((unsigned) clist_max
                                          * sizeof(struct Character));
  if (!clist->ch) {
    printf("Memory Allocation Error in Charlist_Initialize()\n");
    exit(1);
  }
  clist->size = 0;              /* Number of characters */
  clist->allocated = clist_max; /* Size Allocated */
  clist->first = 0;             /* No first character */
  clist->vector_height = vector_height;

  /* Allocate memory for curves and set pointers */
  /* NOTE: To optimize, hardcode the array into the code */
  for (i=0; i<clist_max; i++) {
    clist->ch[i].curve1 = ivector(vector_height);
    clist->ch[i].curve2 = ivector(vector_height);
  }
}

/*-------------------------------------------------------------------*/
/* Zero a previously allocated Characterlist */
void Charlist_zero(clist)
  struct Charlist *clist;
{
  clist->size = 0;              /* Number of characters */
  clist->first = 0;             /* No first character */
}

/*-------------------------------------------------------------------*/
/* Terminate deallocate the Characterlist */
void Charlist_terminate(clist)
  struct Charlist *clist;
{
  int i;

  for (i=clist->allocated-1; i>= 0; i--) { /* Free curves */
    free_ivector(clist->ch[i].curve1);
    free_ivector(clist->ch[i].curve2);
  }
  free(clist->ch);
  clist->allocated = 0;
  clist->first = 0;
}

/*-------------------------------------------------------------------*/
/* Place a new character in the charlist. */
/* Takes the character beyond the charlist [..chr[dist]].
 * calculates .loc, and inserts it into the list */
void Charlist_place_new_char(clist)
  struct Charlist *clist;
{
  struct Character *newchar;    /* Pointer to the new character addhg */

  newchar = &(clist->chr[clist->size]); /* Set newchr */

  /* Calculate the position of the new character */
  Charlist_recalculate_loc(newchar);

  /* Check for full list */
  (clist->size)++;              /* Increment the size of the list */
  if (clist->size >= clist->allocated) /* If too big.. */
  {
    printf("Error in Charlist_place_new_char(): Character list is FULL!\n");
    exit(1);
  }

  /* Place new in the list character */
  Charlist_locate_character(clist, newchar);
}

/*-------------------------------------------------------------------*/
/* This routine repositions a character in the charlist.  It removes
 * the character, recalculates loc and inserts it in the list accordingly
 */
void Charlist_replace_char(clist, ch)
    struct Charlist *clist;
    struct Character *ch;
{
  /* Recalculate the position of the character */
  Charlist_recalculate_loc(ch);

  if (clist->size < 2)          /* If it's the only item in the list */
    return;

  /* Remove the character from the list */
  if (ch->prev == 0) {          /* If 1st on list */
    clist->first = clist->first->next;
    clist->first->prev = 0;
  } else {
    ch->prev->next = ch->next;
    if (ch->next != 0)
      ch->next->prev = ch->prev;
  }

  /* Place the character in the list */
  Charlist_locate_character(clist, ch);
}

/*-------------------------------------------------------------------*/
/* Insert a character after doing a cut */
```

【図222】

```
/* This routine will insert the character beyond the charlist (.chelsize),
 * .loc is recalculated for both ch_after and the new character, however
 * .loc is NOT USED TO PLACE THE CHARACTER.  It will
 * place the new character after ch_after!  After finished, it is an
 * extremely good idea to resort the breaklist!
 */
void Charlist_insert_after_cut(clist, ch_after)
  struct Charlist *clist;
  struct Character *ch_after;   /* Place the new character after this one */
{
  struct Character *newchar;    /* Pointer to the new character adding */

  newchar = &(clist->chr[clist->size]);/* Set newchar */

  /* Calculate the position of the new character */
  Charlist_recalculate_loc(newchar);
  Charlist_recalculate_loc(ch_after);

  /* Check for full list */
  (clist->size)++;              /* Increment the size of the list */
  if (clist->size >= clist->allocated) /* If too big.. */
    {
      printf("Error in Charlist_place_new_char(): Character list is FULL!\n");
      exit(1);
    }

  newchar->prev = ch_after;
  newchar->next = ch_after->next;
  ch_after->next = newchar;
  if (newchar->next != 0)
    newchar->next->prev = newchar;
}

/*.....................................................................*/
/* This routine will remove a character from the character list
 * Note: The memory is NOT reclaimed!!!!  So don't rely on this too much.
 * Also, prev and next pointers of ch are NOT altered !!
 */
void Charlist_remove_char(clist, ch)
   struct Charlist *clist;
   struct Character *ch;
{
  /* Remove the character from the list */
  if (ch->prev == 0)            /* If 1st on list */
    clist->first = clist->first->next;
    clist->first->prev = 0;
  } else {
    ch->prev->next = ch->next;
    if (ch->next != 0)
      ch->next->prev = ch->prev;
  }
}

/*.....................................................................*/
/* This routine will recalculate the position of a character */
void Charlist_recalculate_loc(ch) /* Output is the position */
   struct Character *ch;        /* The character */
{
  int i;loc;
  /* Calculate the position of the new character */
  loc = 0;
  for (i=ch->top; i<ch->bot; i++) /* Position = average position... */
    loc += ch->curve[i] + ch->curve[i+1] /* .. of the curves */
  loc = loc * ( / (ch->bot - ch->top);
                /* Set it */
  ch->loc = loc;
}

/*.....................................................................*/
/* This routine will place a character (that is not in the charlist)
 * into the charlist. */
void Charlist_locate_character(clist, ch)
   struct Charlist *clist;
   struct Character *ch;        /* The character */
{
  int loc;
  struct Character *p;          /* Used for searching for location */
  struct Character *last_p;     /* The character before p */

  /* Is the list empty? */
  if (clist->size == 1)
  {
    ch->next = 0;
    ch->prev = 0;
    clist->first = ch;
    return;
  }

  /* Possible optimization: Don't start from start of list...
     start from where at */
  loc = ch->loc;                /* Retrieve position */

  /* Is it at the beginning of the list? */
  p = clist->first;             /* Load p with the first item on the list */
  if (loc < p->loc)             /* If it's the first on the list */
  {
    p->prev = ch;
    ch->next = clist->first;    /* Set the pointers */
    ch->prev = 0;
    clist->first = ch;          /* Set the 1st on the list to this */
    return;
  }

  while (p->loc <= loc)         /* until find record past where to place */
  {
    last_p = p;                 /* Remember last position */
    p = p->next;                /* Get the next character */
    if (p == 0)                 /* If it's the end of list, insert it there */
      break;
  }
  ch->prev = last_p;
  ch->next = last_p->next;      /* Insert new position after last_p */
  last_p->next = ch;
  if (p!=0)                     /* If neither isn't at the end of the list */
    p->prev = ch;
}

/*.....................................................................*/
/*.....................                                 ...............*/
/*.....................  Zero Vpp Lists Stuff           ...............*/
/*.....................                                 ...............*/
void Zerovpp_Initialize(nalist, size)
   struct NonZeroVpplist *nalist; /* The list */
   int size;                    /* The number of entries (Is allocated!) */
{
  /* Allocate memory for list */
  size = size*2;
  nalist->list = (vector(size); /* Compute memory allocated, etc */
  nalist->allocated = size;     /* Allocate memory */
  nalist->size = 0;             /* Set max allocated */
}
```

【図224】

```
/*
 * Filename: charlist.h
 * Header file for charlist.c
 *
 *
 */

/*****************************************************************/
/* ENUM: Chartype:  Different types of characters */
typedef enum {
    CHARACTER_ERASED,
    CHARACTER_ERASURE_LINE,
    CHARACTER_ERROR_BLIP,
    CHARACTER_TYPED_OUTLINED,
    CHARACTER_COMBINED
} Chartype;

typedef enum {
    PARSEL_RECOGNITION,
    FAILED_RECOGNITION,
    UNRECOGNIZED
} Charrecognition;                  /* Recognition hasn't been tried on it */

/*****************************************************************/
/* Struct Character:
 * Contains information about an outlined character in a page
 */
struct character {
    Chartype type;                  /* The type of character */
    Charrecognition rec;            /* If it's recognized */
    char shell_value;               /* The letter it's recognized as */
    int level_out_at;               /* The level the character was cut at
                                       (for combos) */
    int loc;                        /* Location of the character */
    int top;                        /* The start of the character (top)
                                       (inclusive) */
    int bot;                        /* The bottom of the character (bottom)
                                       (exclusive) */
    int left;                       /* The left most pixel in this character */
    int right;                      /* The right most pixel in this character */
    int *curved;                    /* The left side of the character (inclusive)
                                       (-1 = not there) */
    int *curved2;                   /* The right side of the character (exclusive)
                                       (-1 = not there) */
    struct Character *next;         /* Ptr to next character in the list
                                       (or 0 if last character) */
    struct Character *prev;         /* Ptr to previous character in the list
                                       (or 0 if first character) */
};

/*****************************************************************/
/* Struct Charlist:
 * Contains information about the list of characters */
struct Charlist {
    struct Character *ch;           /* Array: The characters */
    int size;                       /* Number of characters in the ch[] */
    int allocated;                  /* Number of characters allocated */
    int vector_height;              /* The height of the character curves */
    struct Character *first;        /* The first character in linked list */
};

/*****************************************************************/
/*                    Zero Vpp Lists Stuff                       */
/*****************************************************************/
/* Struct MonteroVpplist:
 * Contains information about a list for non-zero VPP's */
struct MonteroVpplist {
    int* list;                      /* Array: 2a - start, 2a+1 = stop */
    int allocated;                  /* Memory allocated (2,4,6...) */
    int size;                       /* How many used (2,4,6...) */
};

/*****************************************************************/
/*                        Define Routines                        */
/*****************************************************************/

#ifdef CPP

/* Define C++ Routines */
extern "C" {
    void Charlist_initialize(struct Charlist *clist, int clist_max,
                             int vector_height);
    void Charlist_terminate(struct Charlist *clist);
    void Charlist_clear(struct Charlist *clist);
    void Charlist_replace_char(struct Charlist *clist, struct Character *ch);
    void Charlist_insert_after_OK(struct Charlist *clist,
                                  struct Character *ch_after);
    void Charlist_remove_char(struct Charlist *clist, struct Character *ch);
    void Charlist_recalculate_loc(struct Character *ch);
    void Zerovpp_initialize(struct MonteroVpplist *zlist, int size);
    void Zerovpp_terminate(struct MonteroVpplist *zlist);
    void Zerovpp_add(struct MonteroVpplist *zlist, int a, int b);
}

#else

/* Define C Routines */
void Charlist_initialize();
void Charlist_zero();
void Charlist_terminate();
void Charlist_replace_new_char();
void Charlist_replace_char();
void Charlist_insert_after_OK();
void Charlist_remove_char();
void Charlist_recalculate_loc();
void Charlist_locate_character();
void Zerovpp_initialize();
void Zerovpp_terminate();
void Zerovpp_add();

#endif
```

```
//
//  Filename:  cimages.cpp
//  This file contains a vast array image utility functions, for a cimage
//
//////////////////////////////////////////////////////////////////////////
//

// undefine this to make images.C run invisibally (except for error)
//#define PRINT_IMAGES_INFO

#include"cimages.h"
#include"xs.h"

//*************************************************************************
//*** CLASS: cimage   (an array of chars)
//*************************************************************************

cimage::cimage()
{
  sizeR = 0;
  sizeC = 0;
  allocated_sizeR = 0;
  allocated_sizeC = 0;
  dpi = 0;
}

cimage::cimage(int r, int c)
{
  sizeR = 0;                    // So problems won't be caused when
  sizeC = 0;                    //   allocate called deallocate
  allocated_sizeR = 0;
  allocated_sizeC = 0;
  dpi = 0;
  allocate(r,c);
}

cimage::cimage(cimage& img)
{
  allocate(sizeR, sizeC); // Allocate memory for new cimage (loads sizes)
  for (int i=0; i<sizeR; i++)      // Copy old image to new cimage
    for (int j=0; j<sizeC; j++)
      pixel[i][j] = img.pixel[i][j];
}


//////////////////////////////////////////////////////////////////////////
// Delete the cimage
cimage::~cimage()
{
  deallocate();
}


//////////////////////////////////////////////////////////////////////////
// Allocate memory for an cimage from memory
void cimage::allocate(int r, int c)
{
  deallocate();                 // Remove current cimage
  if (r < 0 || c < 0)
  {
    cerr << "Illegal cimage size: " << r << " by " << c << "\n";
    exit(1);
  }
  sizeR = r;                            // Set cimage size
  sizeC = c;
  if ((r != 0) && (c != 0))
  {
    pixel = (unsigned char **)malloc((unsigned int) sizeR*sizeof(unsigned char*)); //
Allocate memory
    if (pixel == NULL)    // Allocate memory
    {
      cerr << "Error: Memory Allocation Error (out of memory)\n";
      exit(1);
    }
    for (int i = 0; i<sizeR; i++)
    {
      pixel[i] = (unsigned char *)malloc((unsigned int) sizeC * sizeof(unsigned char)
); // Allocate more mem
      if (pixel[i] == NULL)
      {
        cerr << "Error: Memory Allocation Error (out of memory)\n";
        exit(1);
      }
    }
  }
  allocated_sizeR = sizeR;
  allocated_sizeC = sizeC;
}


//////////////////////////////////////////////////////////////////////////
// Deallocate memory for an cimage from memory
void cimage::deallocate()
{
  if ((allocated_sizeR != 0) && (allocated_sizeC != 0))
  {
    for (int i = 0; i < allocated_sizeR; i++)
      free((char *) pixel[i]);
    if (sizeC != 0)
      free((char **) pixel);
    sizeR = 0;                          // Zero out the sizes
    sizeC = 0;
    allocated_sizeR = 0;
    allocated_sizeC = 0;
  }
}


//////////////////////////////////////////////////////////////////////////
// This allows allocation of memory from outside the class.  It will
// Allocate memory if it's larger.  If not, it will just adjust the size
void cimage::reallocate(int R, int C)
{
  int new_R, new_C;
  if ((R > allocated_sizeR) || (C > allocated_sizeC))
  {
    new_R = allocated_sizeR;
    new_C = allocated_sizeC;
    if (R > new_R)
      new_R = R;
    if (C > new_C)
      new_C = C;
    allocate(new_R,new_C);      // Re-allocate the image
  }
  sizeR = R;
  sizeC = C;
}


//////////////////////////////////////////////////////////////////////////
// Make a cimage with a single value: value = value.  X and Y are the size
void cimage::fill(char value, int R, int C)
{
```

【図226】

```
allocate(R,C);                  // Make the new image (size size adjusted)
for (int i=0; i<size; i++)
  for (int j=0; j<size; j++)
    pixel[i][j] = value;
}

////////////////////////////////////////////////////////////////////////
// Copy a section of the image another.  Image = reference image
// Range is R1,R2 (C1,C2)
void clsImage::clip(clsImage& Image, int R1, int R2, int C1, int C2)
{
  int r,c,t;

  if (R2 < R1)                  // Swap R1 & R2
    { t = R1; R1=R2; R2=t;}
  if (C2 < C1)                  // Swap C1 & C2
    { t = C1; C1=C2; C2=t;}
  if (R1 < 0) R1 = 0;           // Check limits
  if (R2 < 0) R2 = 0;
  if (C1 < 0) C1 = 0;
  if (C2 < 0) C2 = 0;
  if (R1 > Image.size) R1 = Image.size;
  if (R2 > Image.size) R2 = Image.size;
  if (C1 > Image.size) C1 = Image.size;
  if (C2 > Image.size) C2 = Image.size;

  allocate(R2-R1, C2-C1);

  dpt = Image.dpi;
  for (r=0; r<=R2-R1; r++)
    for (c=0; c<=C2-C1; c++)
      pixel[r][c] = Image.pixel[r+R1][c+C1];
}

////////////////////////////////////////////////////////////////////////
// clipspace: reduce the size of an image by removing the blank
// space around it.  This adjusts the image size.
void clsImage::clipspace()
{
  int R1, R2, C1, C2;  // The starting and stopping rows of new image
  int i, j;
  struct clImage temp; // Temporary image

  R1 = size;           // Initialize to extremes
  R2 = 0;
  C1 = size;
  C2 = 0;

  for (i=0; i<size; i++)        // Compute dimensions of new image
    for (j=0; j<size; j++)
      if (pixel[i][j] != 0)
        {
          if (i < R1) R1 = i;   // Left edge?
          if (i > R2) R2 = i;   // Right edge?
          if (j < C1) C1 = j;   // Top edge?
          if (j > C2) C2 = j;   // Left edge?
        }

  if (R1 > R2) R1 = R2 = 0;     // Check for no pixel found
  if (C1 > C2) C1 = C2 = 0;

  temp.allocate(R2-R1+1, C2-C1+1);      // New image is this size
  for (i=R1; i<=R2; i++)                // Copy clipped image
    for (j=C1; j<=C2; j++)
      temp.pixel[i-R1][j-C1] = pixel[i][j];

allocate(temp.size, temp.size);         // Copy temp image to current image
for (i=0; i<size; i++)
  for (j=0; j<size; j++)
    pixel[i][j] = temp.pixel[i][j];
}

////////////////////////////////////////////////////////////////////////
// Read in part of a TIFF file
void clImage::readdiff(char *filename)
{
  FILE *tagfile;                // Input file
  long int tagfile_position;    // The our file position (in bytes)
  long int width;               // Width of picture
  long int length;              // Length of picture
  long int black;               // Value of Black pixel
  long int spp;                 // Bytes per strip
  long int number_of_ifent;     // Start of image data
  long int i,j;
  long int tag, type, len, value;       // Tag values

  if ((tagfile = fopen(filename, "rb")) == NULL)
  {
    cout << "\nError in opening " << filename << "\n";
    exit(1);
  }
  tagfile_position = 0;

  // Read in header and make sure it's Intel
  i = getbytes(tagfile,tagfile_position);
  if (i != 0x4949)
  {
    cout << "... Error: TIFF file not in Intel Format\n";
    exit(1);
  }

  // Read in Magic number
  i = getbytes(tagfile,tagfile_position);
  if (i != 0x002A)
  {
    cout << " BAD Magic Number for TIFF file!!!\n";
    exit(1);
  }

  // Find first file directory
  j = getbytes(tagfile,tagfile_position);
  while(ifent[j])
  {
    i = getbytes(tagfile,tagfile_position);
  }

  // Tags in Directory
  number = getbytes(tagfile,tagfile_position);

  while (--number >= 0) {
    tag = getbytes(tagfile, tagfile_position);
    type = getbytes(tagfile, tagfile_position);
    len = getbyte(tagfile, tagfile_position);
    value = getbytes(tagfile, tagfile_position);

    if (((len & ~ 1) && ((tag & 32768) == 0))
    {
      cout << "Error: Tag length != 1\n";
      exit(1);
```

【図227】

```
while(tagfile.position() < raster_offset
{
  i = getbyte(tagfile.tagfile_position)
}

//.....  READ IN FILE  .....
allocate((int) length, (int) width);

int v;

for (i = 0; i<length; i++)
{
  for (j = 0; j<width; j++)
  {
    if ((j%8) == 0)
    {
      v = getc(tagfile);
      tagfile_position++;
      if (v == EOF)
      {
        cerr << "\n Error: Premature End of File found in diff file\n";
        exit(1);
      }
    }
    if ((v & 128) == black*128)
      pixel[i][j] = 0;
    else
      pixel[i][j] = 1;
    v = v << 1;
  }
}

fclose(tagfile);
}

////////////////////////////////////////////////////////////////////////
// Read in a byte.  File position is updated
// eoferror = 1 to error on EOF, 0 not
int image::getbyte(FILE *f, long int& file_position, int eoferror)
{
  int c1;
  c1 = getc(f);
  if ((c1 == EOF) && (eoferror == 1))
  {
    cerr << "\n Error: Premature End of File in getbyte()\n";
    exit(1);
  }
  file_position++;
  return c1;
}

////////////////////////////////////////////////////////////////////////
// Read in an integer from a file.  (LSB, MSB)
long int image::getbyte(FILE *f, long int& fp)
{
  int c1, c2;
  c1 = getbyte(f, fp, 1);
  c2 = getbyte(f, fp, 1);
  return c1 + c2*256;
}

////////////////////////////////////////////////////////////////////////
// Read in a long integer
long int image::getbyte(FILE *f, long int& fp)
{
  long int c1, c2;

switch (tag)
{
  case 0x00fe:          // NewSubFileType
    if (value != 0)
    {
      cout << "Can't Deal with Non-Standard NewSubFileType\n";
      exit(1);
    }
    break;
  case 0x00ff:          // NewSubFileType
    if (value != 1)
    {
      cout << "SubFile Type not full resolution\n";
      exit(1);
    }
    break;
  case 0x0100:          // Width
    width = value;
    break;
  case 0x0101:          // Length
    length = value;
    break;
  case 0x0102:          // Bits per sample
    if (value != 1)
    {
      cout << " Error, Bits/Sample must be 1!";
      exit(1);
    }
    break;
  case 0x0103:          // Compression
    if (value != 1)
    {
      cout << "Compression Need Error!\n";
      exit(1);
    }
    break;
  case 0x0106:          // Black is
    black = value;
    break;
  case 0x0107:          // Thresholding cag
    if ((value < 1) || (value > 3))
    {
      cout << "Thresholding fag - Illegal value!\n";
      exit(1);
    }
    break;
  case 0x011l:          // Offset to Raster Data
    raster_offset = value;
    break;
  case 0x0117:
    bps = value;
    break;
  case 0x011a:
    break;
  case 0x011b:
    break;
  case 0x0128:
    break;
  default:
    if ((value & 32768) == 0) {
      cout << "!!!!! Warning! An unsupported tag appears in the Tif file!\n";
    }
    break;
  } // switch
} // while
```

【図229】

```
//
//  Filename:   cimages.h
//  This is the header file for cimages.C
//
//////////////////////////////////////////////////////////////////////
//
#define KEN_DEBUG 1
#include<iostream.h>
#include<stdio.h>
#include<stdlib.h>
#include<math.h>

class cimage;           // Forward reference

//---------------------------------------------------------------------
//*** CLASS: cimage
//---------------------------------------------------------------------
class cimage
{
public:
  int           sizeR;          // Number of rows (first index of array)
  int           sizeC;          // Number of cols. (second index of array)
  int           allocated_sizeR; // Amount malloced
  int           allocated_sizeC; // Amount malloced
  int           dpi;            // Dots per inch of the image (0-if unknown)
  unsigned char **pixel;        // The binary image (2d array)

  cimage();                     // Initialize
  ~cimage();                    // Destroy
  cimage(int r, int c);         // Initialize with a size
  cimage(cimage& temp);

  void reallocate(int R, int C); // Will allocate memory for an image
  void fill(char value, int R, int C); // Fill image with a constant value
  void clip(cimage& image, int R1, int R2, int C1, int C2);
  void clipspace();                 // Clips zero space surrounding image
  void readtiff(char *filename);  // Read in a tiff image
  void plot(int x, int y);          // Plot binary image
  void plotbox(int x, int y); // Plot with a box around it

private:
  void allocate(int r, int c);      // Allocate memory
  void deallocate();                // Deallocate memory
  int getibyte(FILE *f, long int& file_position, int &oferror);
  long int get2bytes(FILE *f, long int& fp);
  long int get4bytes(FILE *f, long int& fp);
};
```

【図230】

```
//
// Filename: courcut.c
// This routine cuts courier chars (output from level 1)
// level 1 representation
//
////////////////////////////////////////////////////////////////////////
//
#include"charlist.h"
#include"image.h"

/* Define globals */
int COURCUT_AVG_WIDTH;
int COURCUT_DELTA;

////////////////////////////////////////////////////////////////////////
// This routine will take cuts of the non-zero ypp list
void courcut_process(
    image *im,
    struct HomZeroYpplist *value, // A pointer to the old list (input)
    int start_col,                // Merge to proce
    int stop_col)
{
    int cr_start,  cr_stop;   // Region for the new cut
    int last_cut;             // Position of last cut
    int num_cuts;             // The number of cuts to make
    int cut_value, cut_at;    // The value and location of the likely cut
    int hist[100];            // Histogram ******* CHANGE 100 *******
    int width;                // Width of the line
    int cut_pos;              // Approx position to cut at
    int cut_no;               // The cut number being processed
    int r,c,i;

    /* Calculate num_cuts */; // The width
    width = ypp_edge(start_col, stop_col);
    if (width > COURCUT_AVG_WIDTH+COURCUT_DELTA)
        num_cuts = 2;         // It's only one character
    else {
      // Note: The +2 then /2 will do the division with rounding
      num_cuts = (int)((float) width / (float) COURCUT_AVG_WIDTH + 0.5);
    printf("courcut block width %d => %d characters\n", width, num_cuts);

    last_cut = start_col;
    for (cut_no=1; cut_no<=num_cuts; cut_no++)
    {
      ////////////////////// Break up the region ////////////////////////
      cut_pos = (width * cut_no) / num_cuts + start_col;
      cr_start = cut_pos - COURCUT_DELTA;
      cr_stop  = cut_pos + COURCUT_DELTA;
      if (cr_start < start_col)  // Make sure it doesn't go off edges
        cr_start = start_col;
      if (cr_stop > stop_col)
        cr_stop = stop_col;

      // Make histogram over this range
      for (col=cr_start; col<=cr_stop; col++)
      {
        hist[col-cr_start] = 0;
        for (r=0; r<line.size; r++)
          if (line.pixel[r][col] == 1)
            (hist[col-cr_start])++;
      }

      // Find minimum in histogram
      cut_value = 9999;
      cut_at = (cr_start+cr_stop)/2;
      for (i=cr_start; i<=cr_stop; i++)
        if (hist[i-cr_start] < cut_value)
        {
          cut_value = hist[i-cr_start];
          cut_at = i;
        }

      // Add the cut and continue
      printf("  Cut at %3d,  (searched %d +/- %3d = (%3d,%3d)\n",
          cut_at-start_col, cut_pos-start_col, COURCUT_DELTA,
          cr_start-start_col, cr_stop-start_col);

      // Add the cut and continue
      ZeroYpp_add(value, last_cut, cut_at)
      last_cut = cut_at;
    }
    ////////////////////// Insert last cut ////////////////////////
    ZeroYpp_add(value, last_cut, stop_col);
}

/*************************** MAIN ****************************/
// This routine will take each piece from the input image and send it
// to be processed by the courier cut routine */
void courclist_cut(
    image *im,       // The image of the line to segment
    struct HomZeroYpplist *oldlist, // A pointer to the old list (input)
    struct HomZeroYpplist *list)    // A pointer to the new list (output)
{
  int i;

  list->size = 0;
  for (i=0; i<oldlist->size; i++)
  {
    courcut_process(im, list, oldlist->list[i], oldlist->list[i+1]);
  }
}
```

【図231】

```
//
// Filename: lineprocess.C
// This routine contains stuff to do character segmentation
//
//////////////////////////////////////////////////////////////
//

#include"charlist.h"
#include"cimages.h"
#include"xs.h"
#include"mymalloc.h"
#include"autoinput.h"

#define PASS_TO_TOSHI           // Call Toshi's routines (but ignore output)

#define PASS_TO_LEVEL_3 // Define when runs level 3
//#define PASS_TO_LEVEL_3  // Define if to run level 3
#define PASS_TO_LEVEL_4 // Call the level 4 routines
//#define COMBINING     // Call the combining routines
//#define DASH_CHARS    // Call the routines to cut dashed characters
//#define PRINT_FINAL_CHARLIST

// Declare external routines
void courrier_cut(cimage& line, struct NonZeroVpplist *oldlist,
            struct NonZeroVpplist *list);
void extract_zero_vpp_list(cimage& line, struct NonZeroVpplist *list);
#ifdef PASS_TO_TOSHI
extern "C" char toshi_recognize(int r, int c, char** image);
#endif

// Define global variables
int cursor_X, cursor_Y; // Screen coordinates

extern int WINDOW_SIZE_X;
extern int WINDOW_SIZE_Y;

extern int Logic_type;          // 0 for none, 1 for OR, 2 for AND-OR

// The extern Variables
cimage Process_line;    // The line being segmented (to not reallocate)

//////////////////////////////////////////////////////////////////////
// Misc Utility funciton
int min(                // Returns minimum of two values
  int a,
  int b)
{
  if (a>b)
    return b;
  return a;
}

//////////////////////////////////////////////////////////////////////
// Print a nonzerovpplist on the screen
int print_nzvpplist(            // Returns height printed
    cimage& line,
    struct NonZeroVpplist *nzlist,
    int x,                      // The position
    int y)
{
  int x_left;           /* Left column in nzvpp list */
  int y_start;          /* The starting y */
  int i,r,c;

  x_left = x-nzlist->list[0];
  y_start = y;

  for (i=0; i<nzlist->size; i+=2)
  {
    if (x_left+i*3+nzlist->list[i+1] > WINDOW_SIZE_X - 20) /* Wrap? */
    {
      x_left = x - i*3 - nzlist->list[i];
      y += line.sizeY+10;
    }
    for (r=0; r<line.sizeX; r++)
      for (c=nzlist->list[i]; c<nzlist->list[i+1]; c++)
        if (line.pixel(r)[c] & 1)
          xs_plotxy(x_left + c + i*3, y + r);
    for (r=0; r<line.sizeX; r++)        // Plot double line
    {
      xs_plotxy(x_left+i*3+nzlist->list[i]+1, y+r);
      xs_plotxy(x_left+i*3+nzlist->list[i+1], y+r);
    }
  }
  return line.sizeX + (y - y_start);
}


//////////////////////////////////////////////////////////////////////
// Print a nonzerovpplist on the screen with a character
int print_nzvpplist_char(       // Returns height printed
    cimage& line,
    struct NonZeroVpplist *nzlist,
    char What_Char[],           // The corresponding characters
    int x,                      // The position
    int y)
{
  int x_left;           /* Left column in nzvpp list */
  int y_start;          /* The starting y */
  char dummy_string[10];
  int i,r,c;

  x_left = x-nzlist->list[0];
  y_start = y;

  for (i=0; i<nzlist->size; i+=2)
  {
    if (x_left+i*3+nzlist->list[i+1] > WINDOW_SIZE_X - 20) /* Wrap? */
    {
      x_left = x - i*3 - nzlist->list[i];
      y += line.sizeX+25;
    }
    for (r=0; r<line.sizeX; r++)
      for (c=nzlist->list[i]; c<nzlist->list[i+1]; c++)
        if (line.pixel(r)[c] & 1)
          xs_plotxy(x_left + c + i*3, y + r);
    for (r=0; r<line.sizeX; r++)        // Plot double line
    {
      xs_plotxy(x_left+i*3+nzlist->list[i]-1, y+r);
      xs_plotxy(x_left+i*3+nzlist->list[i+1], y+r);
    }

    // Plot the string
    if (What_Char[i/2] != 0)
    {
      dummy_string[0] = What_Char[i/2];
      dummy_string[1] = 0;
      xs_string(x_left+i*3+(nzlist->list[i]+nzlist->list[i+1])/2-2,
                y+r+9, dummy_string);
    }
  }
  return line.sizeX + (y - y_start);
}
```

【図232】

```
//.....
//.....
//.....                    MAIN
//.....
//.....
//.....
void process_line(
    charpp input_line)          // The image of the line to segment
{
    charpp *line_to_process;    // The line to process
    struct NewZeroVppList *zeilist; // The nonzero vpp list
    struct NewZeroVppList *colist; // The courier cut list
    char what_char[100];        // Which character is in the NEVPP list

//  No Logic (INSERT ROUTINE HERE IF NEEDED)
    line_to_process = &(input_line);        // The default

    struct NewZeroVppList zeroterminalist;  // *** DELETE ***/
    ZeroVpp_InitialiseZeroTerminalist( 200);
    zelist = &zeroterminalist;      // **** ADD THIS MEMBER!!! *****,

    struct NewZeroVppList coursenlist;  // *** DELETE ***/
    ZeroVpp_InitialiseCourseNlist( 200);
    colist = &coursenlist;          // **** ADD THIS MEMBER!!! *****/

#ifdef VISIBLE
    xs_clr_win();
    cursor_x = 20;          // The screen coordinates
    cursor_y = 10;

    (*line_to_process).plot(cursor_x,cursor_y);
    xs_flush();
    cursor_y += (*line_to_process).size + 10;
#endif

/////////// Insert breaks caused by zero values of vpp
    zelist->size = 0;               // Zero the list
    extract_zero_vpp_list((*line_to_process), zelist);

/////////// Insert breaks from the courier cut routine
    courier_cut((*line_to_process), zelist, colist);

#ifdef PASS_TO_TOSH
    int size_r, size_c;
    int r,c,i;
    char **p;
    int row_top, row_bot;
    int col_left, col_right;

    for (i=0; i<colist->size; i++)
    {
      col_left = colist->list[i];
      col_right = colist->list[i+1];

      // Find top and bot rows (skip over blank lines)
      row_top = row_bot = -1;
      for (r=0; (r < (*line_to_process).size) && (row_top == -1); r++)
        for (c=col_left; c<col_right; c++)
          if (((*line_to_process).pixel[r][c] & 1) == 1)
            row_top = r;
      for (r=(*line_to_process).size-1; (r >= 0) && (row_bot == -1); r--)
        for (c=col_left; c<col_right; c++)
          if (((*line_to_process).pixel[r][c] & 1) == 1)
            row_bot = r;

      // Copy image into matrix p and call tosh_recognise
      if ((row_bot != -1) && (row_top != -1))
      {
        size_r = row_bot - row_top;
        size_c = col_right - col_left;
        p = matrix(size_r, size_c);
        for (r=0; r<size_r; r++)
          for (c=0; c<size_c; c++)
            p[r][c] = (*line_to_process).pixel[r+row_top][c+col_left];
        what_char[1/2] = tosh_recognise(size_r, size_c, p);
        free_matrix(p, size_r);
      }
    }
#endif

#ifdef VISIBLE
    #ifdef PASS_TO_TOSH
      cursor_y -= 10*print_nvppplist_char((*line_to_process).colist,
                    what_char,cursor_x,cursor_y);
    #else
      cursor_y -= 10*print_nvppplist((*line_to_process).colist,cursor_x,cursor_y);
    #endif
    xs_flush();
#endif

printf("Displaying Final Segmentation.  Enter 0 to continue? ");
int dummy;
dummy = auto_input_int();

ZeroVpp_Terminate(&zeroterminalist);    // *** DELETE ****
ZeroVpp_Terminate(&coursenlist);        // *** DELETE ****
}
```

【図233】

```
/* This program contains common vector and matrix routines */
#include<malloc.h>
#include<stdio.h>

/* Routine for error during memory allocation */
void allocation_error(s)
    char* s;                    /* The error text */
{
    int *core=NULL;

    printf("Memory allocation error %s\nProgram Aborted.\n", s);
    *core=0;
    core=NULL = 0;
}

/************ IVECTOR ****************/
/* Allocate an integer vector of size [0..n-1] */
int *ivector(n)
    int n;                      /* Size of vector */
{
    int *v;

    v = (int *)malloc((unsigned) n*sizeof(int));
    if (!v) allocation_error("in ivector()");
    return v;
}

/* Free an integer vector v of size [0..n-1] */
void free_ivector(v)
    int *v;                     /* The vector */
{
    free((int *) v);
}

/************ FVECTOR ****************/
/* Allocate an float vector of size [0..n-1] */
float *fvector(n)
    int n;                      /* Size of vector */
{
    float *v;

    v = (float *)malloc((unsigned) n*sizeof(float));
    if (!v) allocation_error("in fvector()");
    return v;
}

/* Free an float vector v of size [0..n-1] */
void free_fvector(v)
    float *v;                   /* The vector */
{
    free((float *) v);
}

/************ RANGE IVECTOR ****************/
/* Allocate a ranged vector v of size [m..n-1] */
int *ivector_ranged(m,n)
    int m,n;                    /* The range */
{
    int *v;

    v = (int *)malloc((unsigned) (n-m)*sizeof(int));
    if (!v) allocation_error("in ivector_ranged()");
    return (v-m);
}

/* Free an integer vector v of size [n..n-1] */
void free_ivector_ranged(v,n)
    int *v;                     /* The vector */
    int n;                      /* The starting index of v */
{
    free((int *) (v+n));
}

/************ IMATRIX ****************/
/* Allocate an integer matrix of size [0..r-1][0..c-1] */
int **imatrix(r, c)
    int r,c;                    /* The size of the matrix */
{
    int** m;
    int i;

    m=(int **)malloc((unsigned) r * sizeof(int *));
    if (!m) allocation_error("1 in imatrix()");
    for (i=0; i<r; i++) {
      m[i] = (int *)malloc((unsigned) c * sizeof(int));
      if (!m[i]) allocation_error("2 in imatrix()");
    }
    return m;
}

/* Free an integer matrix m of size [0..r-1][0..c-1] */
void free_imatrix(m, r)
    int **m;                    /* The matrix */
    int r;                      /* The size (num rows) */
{
    int i;

    for (i=r-1; i>=0; i--)
      free((int *) m[i]);
    free((int **) m);
}

/************ CMATRIX ****************/
/* Allocate an char matrix of size [0..r-1][0..c-1] */
char **cmatrix(r, c)
    int r,c;                    /* The size of the matrix */
{
    char** m;
    int i;

    m=(char **)malloc((unsigned) r * sizeof(char *));
    if (!m) allocation_error("1 in cmatrix()");
    for (i=0; i<r; i++) {
      m[i] = (char *)malloc((unsigned) c * sizeof(char));
      if (!m[i]) allocation_error("2 in cmatrix()");
    }
    return m;
}

/* Free an character matrix m of size [0..r-1][0..c-1] */
void free_cmatrix(m, r)
    char **m;                   /* The matrix */
    int r;                      /* The size (num rows) */
{
    int i;

    for (i=r-1; i>=0; i--)
      free((char *) m[i]);
    free((char **) m);
}
```

【図236】

```
//
// Filename: pagman.C
// This routine contains stuff to do page segmentation
//
//////////////////////////////////////////////////////////////////////
//......................................................................

#define MAX_CHARS_PER_LINE 250
#define MAX_PIXEL_HEIGHT 150

#include<stdlib.h>
#include<string.h>
#include<math.h>
#include"image.h"
#include"malloc.h"
#include"charlist.h"
#include"autoinput.h"

void process_line(cimage& image);
void segp_init(cimages& page);
void segp_ploc_lines(cimages& page, cimage& line, Charlist *clist, int line_top,
                     int line_bot);

int segp_ploc_page(cimages& page, int start_row);

#define MAXDIVISIONS 15   // The number of columns to divide the page into (Original 12)
#define CLOSE_OVERLAP 0.3     // What fraction of two lines must overlap (originally 0.3)
#define PERP_DPI_3_MIN_BLOCKSIZE 0.034     // Minimum line height (inches)
#define RAD (180.0/3.14159)
#define OVERLAP 1

// Define external functions
extern cimage *process_line;
extern void Rec_load_font(char *filelib); // Loads in the font

// Global Variables for page size (eliminate border noise

int PAGE_R1;              // The starting row of the page (inclusive)
int PAGE_R2;              // The stopping row of the page (exclusive)
int PAGE_C1 = PAGE_R1;    // The starting col of the page (inclusive)
int PAGE_C2;              // The stopping col of the page (exclusive)
int PAGE_CO;              // PAGE_C1 - PAGE_C1

// Define temporary global variables
int WINDOW_SIZE_X;
int WINDOW_SIZE_Y;        // Used for plotting
int cursorX = 10;
int cursorY = 10;         // The screen coordinates

extern int COURCUT_AVG_WIDTH;
extern int COURCUT_ZERO;

//.......... This routine will ploc a scaled cimage on the screen ..........
void clear_scale_and_ploc_cimage(
  cimage& image)          // The image
{
  float scale, scale2;

  int PX, PY, i, c, x, y;
  int i, j, ii, ij, jj, jj;

  // Compute scale to fit image on screen (x and y)
  scale = (float) (WINDOW_SIZE_X-10)/ (float) image.sizeX;
  scale2 = (float) (WINDOW_SIZE_Y-20)/ (float) image.sizeY;

  if (scale2 < scale)          // Take the least of the two
    scale = scale2;
  if (scale > 1.0)             // Only shrink the image
    scale = 1.0;

  PY = (int) ((double) image.sizeY * scale);
  PX = (int) ((double) image.sizeX * scale);

  xs_clr_win();                // Clear the window
  // Draw the image
  for (r=0; r<PY; r++)
    for (c=0; c<PX; c++)
    {
      x = (int) ((double) c / scale);
      y = (int) ((double) r / scale);
      if (image.pixel[y][x] != 0)
        xs_ploty(10+c, 30+r); // Border = 10;
    }

  // Draw Box Around Image
  ii = 0;
  jj = PY + 12;
  for (i=0; i<=MAXDIVISIONS; i++)
  {
    if ((i+1)%2)
      xs_ploty(ii, jj);
    if ((i+2)%2)
      xs_ploty(ii, jj);
    ii = ii + PX/MAXDIVISIONS;
  }
  ii = 0;
  ij = PX + 12;
  for (j=0; j<=PY+12; j++)
  {
    if ((j+3)%2)
      xs_ploty(ii, jj);
    if ((j+2)%2)
      xs_ploty(ii, jj);
  }
  xs_flush();                  // Refresh the window
}

//.....................................................................
// This routine will create a histogram from col n to n.
// THIS WILL INSERT A 1 FOR TEXT 0 FOR BLANK!
void Rec_histogram(
  cimage& image,               // The input image
  int n,                       // The starting and stopping col (inclusive)
  int m,                       // The ending col (inclusive)
  int *a)                      // The output histogram (allocated previously)
{
  int r,c;

  for (r=0; r<PAGE_R; r++)     // Zero before PAGE_R:
    a[r] = 0;
  for (r=PAGE_R1; r<PAGE_R2; r++)     // Histogram between R1 and R2
  {
    a[r] = 0;
```

【図237】

```
for (row; row<...; row++)
  if (page.pixel[r][c] < 0)
    {
      a[r]++;
      c = h;
    }                                   // Exit this search
  }
}
for (r=PAGE_R; r<page.rise; r++)        // Zero after PAGE_R
  a[r] = 0;
}

//.....
// Given a range (a1,a2) and (b1, b2) will compute fraction of union/intercept
// Required: a1 <= a2, and b1 <= b2, and the ranges overlap.
// If not overlap, will give a negative number
float overlap(
  int a1,                               // Range 1
  int a2,
  int b1,                               // Range 2
  int b2)
{
  int i1, i2;                           // The intercept range
  int u1, u2;                           // The union range

  if (a1 < b1) {
    i1 = b1;
    u2 = a2;
  } else {
    i1 = a1;
    u1 = b1;
  }
  if (a2 < b2) {
    i2 = a2;
    u2 = b2;
  } else {
    i2 = b2;
    u1 = a1;
  }
  if (u1 == u2)                         // Only you can prevent floating point exceptions!
    return 0.0;

  return (float) (i2-i1) / (float) (u2-u1);
}

//.....
// This routine will segment an entire page given the para
// It will fill in the division (or each line
// Used PAGE_R1 etc... Note: PAGE_R1 must NOT be zero, and PAGE_R1 must NOT
//     be image.rise.
//
void segment_page(
    class Page
    struct CharStat *clist)     // A characterist (memory allocated only)
{
  class Line;                   // The line, one at a time
  int *startcol;                // The start of a column (inclusive)
  int *stopcol;                 // The stop of a column (exclusive)
  int *hgram;                   // Array of histograms [num] [row]
  int **startcol;               // Array of starting text positions
  int **stopcol;                // Array of stopping positions
  int *current;                 // The current for the start and stop tests
  int *size;                    // The size of each list
  int *include;                 // Include this segment in the next line (0.no, 1.yes)
  int end_of_list;              // Loop termination flag
  int top_div;                  // The division with the topmost text
  int range1, range2;           // The text range in the adjacent division (row 1's)
  int r1, r2, c1, c2;           // The text range in the current division (row 1's)
  int r;                        // The row and column of line in page
  int min_line_height;          // The minimum line height
  int reject;                   // Line rejection flag
  int i,j,a,b,c,r;              // Nice handy variable

  startcol = Ivector(NDIVISIONS);
  stopcol = Ivector(NDIVISIONS);
  hgram = Imatrix(NDIVISIONS, page.rise);   // This is overkill
  startcast = Imatrix(NDIVISIONS, page.rise);   // This is overkill
  stopcast = Imatrix(NDIVISIONS, page.rise);
  current = Ivector(NDIVISIONS);
  size = Ivector(NDIVISIONS);
  include = Ivector(NDIVISIONS);

  min_line_height = (int) ((float) page.dpi * PSEG_DIF_3_MIN_BLOCK_SIZE);

  ////////// Step 1:  Define columns and find histograms //////////
  // Find range of columns
  startcol[0] = PAGE_L1;
  for (i=0; i<NDIVISIONS-1; i++)
  {
    c = PAGE_C1 + (PAGE_C2 - (i-1)) / NDIVISIONS;
    startcol[(i+1] = c - OVERLAP;
    stopcol[i] = c + OVERLAP;
  }
  stopcol[NDIVISIONS-1] = PAGE_C2;
  // Find histograms
  for (i=0; i<NDIVISIONS; i++)          // For each division
  {
    make_partial_histo(page, startcol[i], stopcol[i], hgram[i]);
  }

  ////////////////// Step 2:  Make text area lists ////////////////////
  for (i=0; i<NDIVISIONS; i++)
    // Note, Zeros are guaranteed on both ends of the histogram
  {
    size[i] = 0;                        // Initialize
    for (j=PAGE_R1-1; j<PAGE_R2; j++)
    {
      if ((hgram[i][j] == 0) && (hgram[i][j-1]) != 0)   // If starting text
        startcast[i][size[i]] = j+1;
      if ((hgram[i][j] != 0) && (hgram[i][j+1]) == 0)   // If stopping text
      {
        stopcast[i][size[i]] = j+1;
        size[i] += 1;
      }
    }
  }

  ////////////////// Step 3:  Find the lines ////////////////////
  for (i=0; i<NDIVISIONS; i++)
    current[i] = 0;                     // Set all pointers to the top of the list
  end_of_list = 0;

  while(!end_of_list == 0)
  {
    ////////// Step 3a:  Find the highest text
    a = 2*page.rise;                    // Going to be high_div
    for (i=0; i<NDIVISIONS; i++)
      if (current[i] < size[i])         // If valid cursor
      {
        b = startcast[i][current[i]] - stopcast[i][current[i]];
```

【図238】

```
    if (b < a)                          // If it's better
    {
      a = b;
      top_div = i;
    }
  }

for (i=0; i<NDIVISIONS; i++)            // Initialize include[]
  include[i] = 0;                       // Don't include anything
include[top_div] = 1;                   // Include the first one

r1 = starttext[top_div][cursor[top_div]];   // Initialize the size of the
r2 = stoptext[top_div][cursor[top_div]];    //   cimage to this segment
c1 = startcol[top_div];
c2 = stopcol[top_div];

///////////// Step 3b: Loop to left and see what columns match
range1 = r1;                    // The adjacent division text range
range2 = r2;
for (i=top_div-1; i>=0; i--)           // Each column to the left
{
  if (cursor[i] < size[i])             // If we're past the end, don't do it
  {
    crange1 = starttext[i][cursor[i]];
    crange2 = stoptext[i][cursor[i]];
    if (overlap(range1, range2, crange1, crange2) > CLOSE_OVERLAP)
                            // If overlapping
    {
      include[i] = 1;                          // Include this column also
      if (crange1 < r1)               // Adjust range of  r1, r2, c1, c2
        r1 = crange1;
      if (crange2 > r2)
        r2 = crange2;
      c1 = startcol[i];
      range1 = crange1;               // Use this for the next range
      range2 = crange2;
    }
  }
}

///////////// Step 3c: Loop to right and see what columns match
range1 = starttext[top_div][cursor[top_div]];  // The adjacent division
range2 = stoptext[top_div][cursor[top_div]];   // text range.
for (i=top_div+1; i<NDIVISIONS ; i++)        // Each column to the left
{
  if (cursor[i] < size[i])             // If we're past the end, don't do it
  {
    crange1 = starttext[i][cursor[i]];
    crange2 = stoptext[i][cursor[i]];
    if (overlap(range1, range2, crange1, crange2) > CLOSE_OVERLAP)
                            // If overlapping
    {
      include[i] = 1;                          // Include this column also
      if (crange1 < r1)               // Adjust range of  r1, r2, c1, c2
        r1 = crange1;
      if (crange2 > r2)
        r2 = crange2;
      c2 = stopcol[i];
      range1 = crange1;               // Use this for the next range
      range2 = crange2;
    }
  }
}

///////////// Step 3d: Copy the pieces into the cimage //////////////////
// Optimize this.  Selectively do fill() and clipspace()
    line.fill(0, r2-r1, c2-c1);          // Fill up the image with zeros
    for (i=0; i<NDIVISIONS; i++)
      if (include[i] == 1)
      {
        for (c=startcol[i]; c<stopcol[i]; c++)
          for (r=starttext[i][cursor[i]]; r<stoptext[i][cursor[i]]; r++)
            line.pixel[r-r1][c-c1] = (int) page.pixel[r][c];
      }
     line.clipspace();              // Remove white space around the line

    //////////////// Step 3e: Do we reject this image? //////////////////
    reject = 0;                    // Default is accept
    if ((r2-r1) < min_line_height)
      reject = 1;
//    printf("   cimage range is %d..%d, %d..%d\n", c1, c2, r1, r2);

    //////////////// Step 4: Process Line //////////////////
    if (reject == 0)
    {
#ifdef VISIBLE
      clear_scale_and_plot_cimage(line);
#endif
      printf("Enter 0 to run character segmentation, 1 to ignore? ");
      r = auto_input_int();
      if (r == 0)
      {
        process_line(line);
//        segp_plot_line(page, line, clist, r1, r2);
      }
    }

    //////////////// Skip over all the used sections
    for (i=0; i<NDIVISIONS; i++)           // Up all the cursors
      if (include[i] == 1)
        cursor[i] += 1;                    // Skip over the included areas

    ////////// Check if we've gone off the end of the list
    end_of_list = 1;
    for (i=0; i<NDIVISIONS; i++)
      if (cursor[i] < size[i])             // If the cursor is not at the end...
        end_of_list = 0;                   // ... don't exit

  }

  free_ivector(include);
  free_ivector(size);
  free_ivector(cursor);
  free_imatrix(stoptext, NDIVISIONS);
  free_imatrix(starttext, NDIVISIONS);
  free_imatrix(hgram, NDIVISIONS);
  free_ivector(stopcol);
  free_ivector(startcol);
}

//**************************************************************
//**************************************************************
//*************            MAIN               ****************
//**************************************************************
//**************************************************************
void xa_init()
{
  cimage page;
  struct Charlist clist;
  char filename[100], filename1[100], filename2[100];
```

【図239】

```
  int border_ignore;
  int i;

  Charlist_initialise(&clist, MAX_CHARS_PER_LINE, MAX_LINE_HEIGHT);
  Process_line.fill(0,80,2500); // Allocate the memory for the lines

  xs_win_dim(&WINDOW_SIZE_X, &WINDOW_SIZE_Y);  // Get the window size

  printf("Input tiff filename? /dl/tif/");
  strcpy(filename1, "/dl/tif/toshi1.tif");
  scanf("%s", filename1);
  strcpy(filename, "/dl/tif/");
  strcat(filename, filename1);

  auto_initialise();            // Initialise Auto-Input routines

  printf("Enter courcut average width? ");
  COURCUT_AVG_WIDTH = auto_input_int();

  printf("Enter courcut delta? ");
  COURCUT_DELTA = auto_input_int();

  page.dpi = 100;

  border_ignore = 1;
//  cout << "Input border width to ignore (in pixels)? ";
//  cin >> border_ignore;

  printf("Reading in File... ");
  page.readtiff(filename);
  printf("Done.\n");

//  segp_init(page);                    // Initialise page printing routines

//  border_ignore = (int) ((float) page.dpi * DPI_2_BORDER_IGNORE);
//  printf("Border ignore = %d\n", border_ignore);

  PAGE_R1 = border_ignore;              // Define page size
  PAGE_R2 = page.sizeR - border_ignore;
  PAGE_RD = PAGE_R2 - PAGE_R1;
  PAGE_C1 = border_ignore;
  PAGE_C2 = page.sizeC - border_ignore;
  PAGE_CD = PAGE_C2 - PAGE_C1;


  // ************ THIS IS JUST FOR TESTING ********************
  xs_flush();
  printf("Enter 0 to continue? ");
  i = auto_input_int();

  // clear_scale_and_plot_cimage(page);

  printf("Pausing because page is displayed. Input 0 to segment page? ");
  border_ignore = auto_input_int();

  while(1) {
    segment_page(page, &clist);
    xs_flush();

    printf("Program finished. Enter 0 to run again? ");
    int ttemp;
    ttemp = auto_input_int();
  }

  Charlist_terminate(&clist);
  auto_terminate();
}
```

[図241]

```
/*
FILE: util.c

COMMENTS:
      This file contains general utility routines.

      ut_reverse(width,height,buf);
      ut_getbits(x,p,n);
      ut_ascii_bitmap(width,height,buf);

   Usage:

      Just call the routines.
*/

/* ------------------------Includes------------------------- */
#include <stdio.h>
#include <malloc.h>

/* ------------------------Global--------------------------- */

/* ------------------------Externs-------------------------- */

/*
Routine: ut_reverse(width,height,buf)

Description: Reverse the bits for display format

Return: Nothing

Author: Rick Lee
*/
ut_reverse(width,height,buf)
int width;
int height;
char *buf;
{

   unsigned char val,new_val;
   int x,z,y;

   for(x = 0; x < (width>>3)*height; x++)
   {
      z = 0;
      val = buf[x];
      new_val = 0;
      for(y = 7; y >= 0 ; y--)
      {
         if(ut_getbits(val,z++,1))
            new_val = 0x01 << y | new_val;
      }
      buf[x] = new_val;
   }
}


/*
Routine: ut_getbits(x,p,n)

Description: Check if the bit is turned on or not

Return: Nothing

Author: Rick Lee
*/
ut_getbits(x,p,n)
unsigned char x;     /* The character to slice bit(s) from */
int p;               /* The begining of the bit number */
int n;               /* Number of bits to get */
{
   return(x >> (p+1-n)) & ~(~0 << n);
}

/*
Routine: ut_ascii_bitmap(width,height,buf)

Description: Display the bitmap using ascii characters

Return: Nothing

Author: Rick Lee
*/
ut_ascii_bitmap(width,height,buf)
int width;
int height;
char *buf;
{
   short bit_code = 0x0080;
   short bit_result,t=0;
   short x,y,z,bits;

   for(x = 0; x < height; x++)
   {
      for(y = 0; y < width; y++)
      {
         bits = buf[t++];
         for(z = 0; z < 8; z++)
         {
            bit_result = bits & bit_code;
            if(bit_result != 0)
               printf("1");
            else
               printf("0");
            bit_code = bit_code >> 1;
         }
         bit_code = 0x0080;
      }
      printf("\n");
   }
}
```

[図242]

```
/*
FILE: xs.C

COMMENTS:
      This file contains routines for displaying 'things'.

      xs_flush();
      xs_plotxy(x,y);
      xs_unplotxy(x,y);
      xs_redisplay();
      xs_dis_bitmap(x,y,width,height,buf);
      xs_clr_win();
      xs_draw_box(x,y,width,height);
      xs_string(x,y,str);
      xs_setcolor(color);
      xs_win_dim(width,height);

   Usage:

      Just call these routines once the hook has been made with the
      display routine.
*/

/* ----------------------Includes---------------------- */
#include <stdio.h>
#include <stdlib.h>
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <Xm/Xm.h>
#include <Xm/DrawingA.h>
#include <Xm/PushB.h>
#include <Xm/Form.h>
#include <X11/Xutil.h>

/* ----------------------Global Var---------------------- */



/* ----------------------Externs---------------------- */
extern Widget parent,           /* The root widget */
      canvas;                   /* The display widget */
extern GC line_gc;
extern XGCValues gcv;

xs_flush()
{
   /* Flush the queue to X-server */
   XFlush(XtDisplay(canvas));
}


/*
Routine: xs_plotxy(x,y)

Description: Draw a point on the screen

Return: Nothing

Author: Rick Lee
*/
xs_plotxy(x,y)
int x;
int y;
{
   XDrawPoint(XtDisplay(canvas),XtWindow(canvas),line_gc,x,y);
}

/*
Routine: xs_unplotxy(x,y)

Description: UNDraw a point on the screen

Return: Nothing

Author: Rick Lee
*/
xs_unplotxy(x,y)
int x;
int y;
{
   XSetForeground(XtDisplay(canvas),line_gc,gcv.background);
   XDrawPoint(XtDisplay(canvas),XtWindow(canvas),line_gc,x,y);
   XSetForeground(XtDisplay(canvas),line_gc,gcv.foreground);
}


/*
Routine: xs_redisplay()

Description: Redisplay the 'stuff' that was on the window

Return: Nothing

Author: Rick Lee
*/
xs_redisplay()
{
}


/*
Routine: xs_dis_bitmap(x,y,width,height,buf)

Description: Draw the given bitmap

Return: Nothing

Author: Rick Lee
*/
xs_dis_bitmap(x,y,width,height,buf)
{

   Pixmap bitmap,pix;   /* The bitmap for display */

   /* Reverse the bit ordering */
   ut_reverse(width,height,buf);

   /* First create bitmap */
   bitmap = XCreateBitmapFromData(XtDisplay(canvas),
                                  RootWindowOfScreen(XtScreen(canvas)),
                                  buf,width,height);

   /* Create the pixmap area */
   pix = XCreatePixmap(XtDisplay(canvas),RootWindowOfScreen(XtScreen(canvas)),width,h
eight,
                       DefaultDepthOfScreen(XtScreen(canvas)));


   /* Now copy the data to bitmap */
   XCopyPlane(XtDisplay(canvas),bitmap,pix,line_gc,0,0,
              width,height,0,0,1);

   /* Now copy the pixmap to drawing area */
```

【図243】

```
    XCopyArea(XtDisplay(canvas),pix,XtWindow(canvas),line_gc,0,0,
              width,height,x,y);

    /* Now free the bitmap */
    XFreePixmap(XtDisplay(canvas),bitmap);

    /*  Now free the bitmap */
    XFreePixmap(XtDisplay(canvas),pix);

    /* Now flush the queue to X-server */
    XFlush(XtDisplay(canvas));
}



/*
Routine: xs_clr_win()

Description: Clear the screen

Return: Nothing

Author: Rick Lee
*/
xs_clr_win()
{
   XClearArea(XtDisplay(canvas),XtWindow(canvas),0,0,0,0,TRUE);
}


/*
Routine: xs_draw_box(x,y,width,height)

Description: Draw a box on screen

Return: Nothing

Author: Rick Lee
*/
xs_draw_box(x,y,width,height)
int x;
int y;
int width;
int height;
{
   XDrawRectangle(XtDisplay(canvas),XtWindow(canvas),line_gc,
                  x,y,width,height);
}


/*
Routine: xs_string

Description: Draw a string on the screen

Return: Nothing

Author: Rick Lee
*/
xs_string(x,y,str)
int x;
int y;
char *str;
{
   XDrawString(XtDisplay(canvas),XtWindow(canvas),line_gc,
               x,y,str,strlen(str));
}
```

```
/*
Routine: xs_setcolor

Description: Set the current color

Return: Nothing

Author: Rick Lee
*/
xs_setcolor(color)
int color;
{
   gcv.foreground = color;
   XSetForeground(XtDisplay(canvas),line_gc,color);
}

/*
Routine: xs_win_dim

Description: Get the size of the window

Return: Nothing

Author: Rick Lee
*/
xs_win_dim(width,height)
int *width;
int *height;
{
   Arg args1[2];
   Dimension win_width,win_height;

   XtSetArg(args1[0],XtNwidth,&win_width);
   XtSetArg(args1[1],XtNheight,&win_height);
   XtGetValues(canvas,args1,2);

   *width = win_width;
   *height = win_height;
}

/*
xs_set_cmap(w)
Widget w;
{
   int ncolors;
   XColor colors[256];
   Colormap my_cmap;
   int count = 1;


   Display *d;
   int scr;
   Colormap def_colormap;
   XColor Color;
   int x;
   int the_color = 2080;

   d = XtDisplay(w);
   scr = DefaultScreen(d);
   ncolors = DisplayCells(d,scr);
   def_colormap = DefaultColormap(d,scr);
   for(x = 0; x < ncolors; x++)
   {
      colors[x].pixel = x;
```

[図246]

```
/*
FILE: xt.C

COMMENTS:
        This file contains interface to X-display routines.

        main();
        xt_redisplay(w,data,call_data);

    Usage:
        Your initilize routine must be xs_init().
*/

/* ---------------------Includes---------------------- */
#include <stdio.h>
#include <stdlib.h>
#include <X11/IntrinsicP.h>
#include <X11/StringDefs.h>
#include <Xm/Xm.h>
#include <Xm/DrawingA.h>
#include <Xm/PushB.h>
#include <Xm/Form.h>
#include <X11/Xutil.h>

/* ---------------------Global Var---------------------- */
Widget parent,              /* The root widget */
       canvas;              /* The widget to display */
int start_up = 1;           /* The flag to call init routine */
GC line_gc;                 /* Pointer to GC */
XGCValues gcv;              /* The GC values */

/* ---------------------Externs---------------------- */



/*
Routine: xt_redisplay(w,data,call_data)

Description: Expose callback

Return: Nothing

Author: Rick Lee
*/
void xt_redisplay(w,data,call_data)
Widget w;                           /* The widget associated with display */
char *data;                         /* The data  */
XmDrawingAreaCallbackStruct *call_data;   /* Not used */
{
   if(start_up)
   {
      start_up = 0;
      xs_init();
   }
}

main(argc,argv)
int argc;
char *argv[];
{
   Arg args[10];                    /* Arguments to X-server */
   int n=0;                         /* Number of arguments */
   int background,                  /* Default background and foreground color */
       foreground;

#ifdef CPP
   /* Call the routine to setup C++ interface. take out if no C++ interface */
   _main();
#endif

   /* Init the Xt function */
   parent = XtInitialize(argv[0],"Pixmap",NULL,0,&argc,argv);


   /* Create drawing area widget */
   n=0;
   canvas = XtCreateManagedWidget("canvas",xmDrawingAreaWidgetClass,
                                  parent,args,n);

   /* Get the background and foreground color */
   XtSetArg(args[0],XtNforeground,&foreground);
   XtSetArg(args[1],XtNbackground,&background);
   XtGetValues(canvas,args,2);

   /* Now add callbacks and event handler */
   XtAddCallback(canvas,XmNexposeCallback,xt_redisplay,NULL);

   /* Display the widget and enter event loop */
   XtRealizeWidget(parent);

   /* Set the GC for display routines */
   gcv.foreground = foreground;
   gcv.background = background;
   line_gc = XCreateGC(XtDisplay(canvas),
                       DefaultRootWindow(XtDisplay(canvas)),
                       GCForeground | GCBackground,&gcv);

   XtMainLoop();
}
```

【図247】

```
//
// Filename: zerovpp.C
// This routine contains the lines to find Blocks that have non-zero vpp
// (level 1 segmentation)
//
////////////////////////////////////////////////////////////////////////
//
#include"charlist.h"
#include"cimager.h"

////////////////////////////////////////////////////////////////////////
// Scan a line quickly (every 1/4 pixel) looking for an 'on' pixel
int line_scan_quick(    // Returns 1 for pixel found, 0 otherwise
  cimage& line,         // The line to process
  int c)                // The column to scan
{
  int r;

  for (r=0; r<line.sizeR; r+=4)
    if (line.pixel[r][c] & 1)
      return 1;
  return 0;
}


////////////////////////////////////////////////////////////////////////
// Scan a line completely looking for an 'on' pixel
int line_scan_complete( // Returns 1 for pixel found, 0 for otherwise
  cimage& line,         // The line
  int c)                // The column to scana
{
  int r;

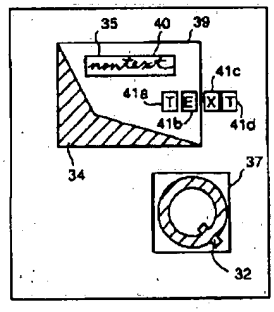
  for (r=0; r<line.sizeR; r++)
    if (line.pixel[r][c] & 1)
      return 1;
  return 0;
}


////////////////////////////////////////////////////////////////////////
// The main routine.  Call this to make the list
void extract_zero_vpp_list(
  cimage& line,         // The image of the line to segment
  struct MonteroVppList *list)  // A pointer to the list
{
  int c;
  int left, right;

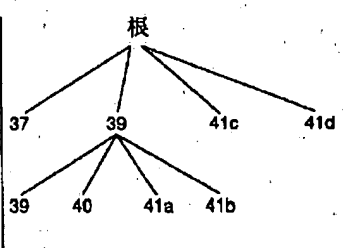
  for (c=0; c<line.sizeC; c+=4)  // Scan the image
  {
    if (line_scan_quick(line, c))  // If found something
    {
      left = right = c;          // Look to the left for the start
      while (--c >= 0)
        if (line_scan_complete(line, c) == 0)
          break;
      left = c+1;
      c = right;                 // Keep looking to the right
      while (++c < line.sizeC)
        if (line_scan_complete(line, c) == 0)
          break;
      right = c;
      zerovpp_add(list, left, right);   // Add in the new break
    }
  }
```

【手続補正書】
【提出日】平成5年5月13日
【手続補正1】
【補正対象書類名】図面
【補正対象項目名】全図
【補正方法】変更
【補正内容】

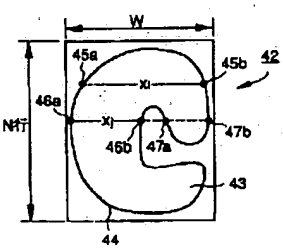
【図5B】



【図5C】



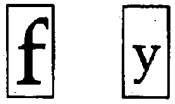
【図6A】



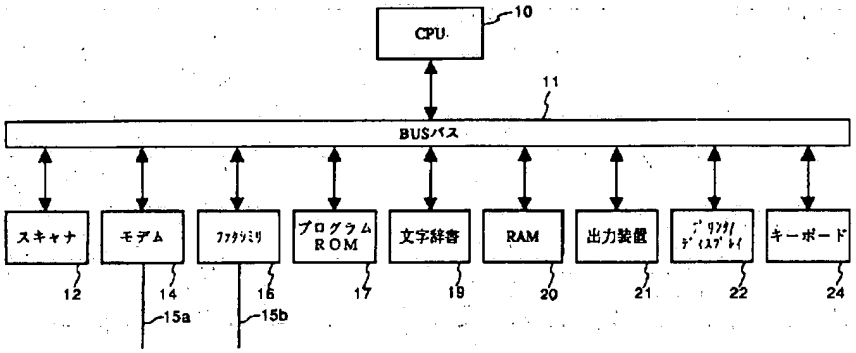
【図26B】【図26C】



【図26D】



【図1】　＊　＊

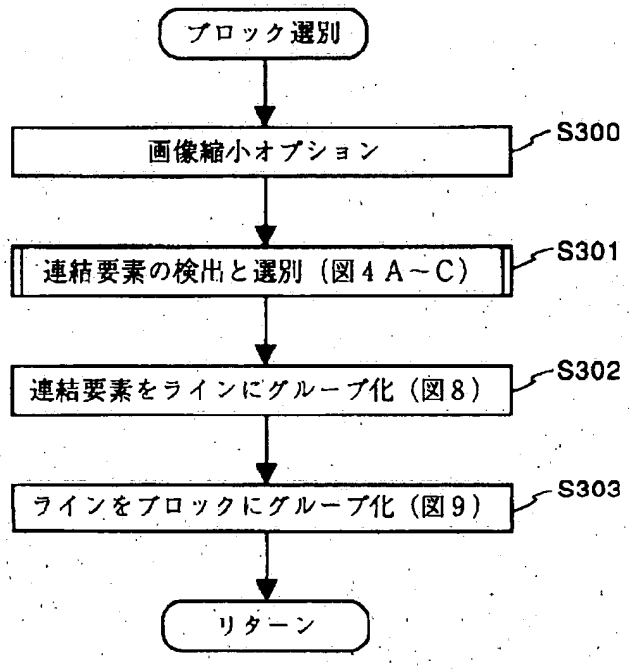


【図35】

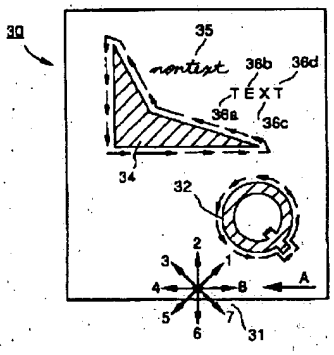
Source Code For

Block Selection

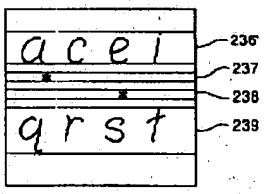
【図3】　※　※



【図5A】



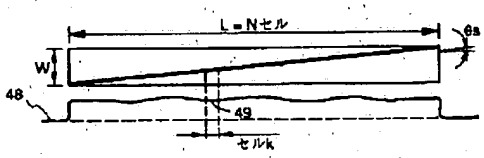
【図18B】



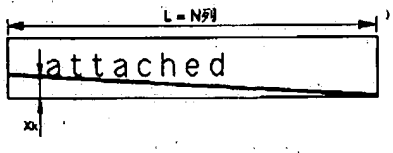
【図234】

```
freei(char **) m);
}
```

【図6B】



【図6C】



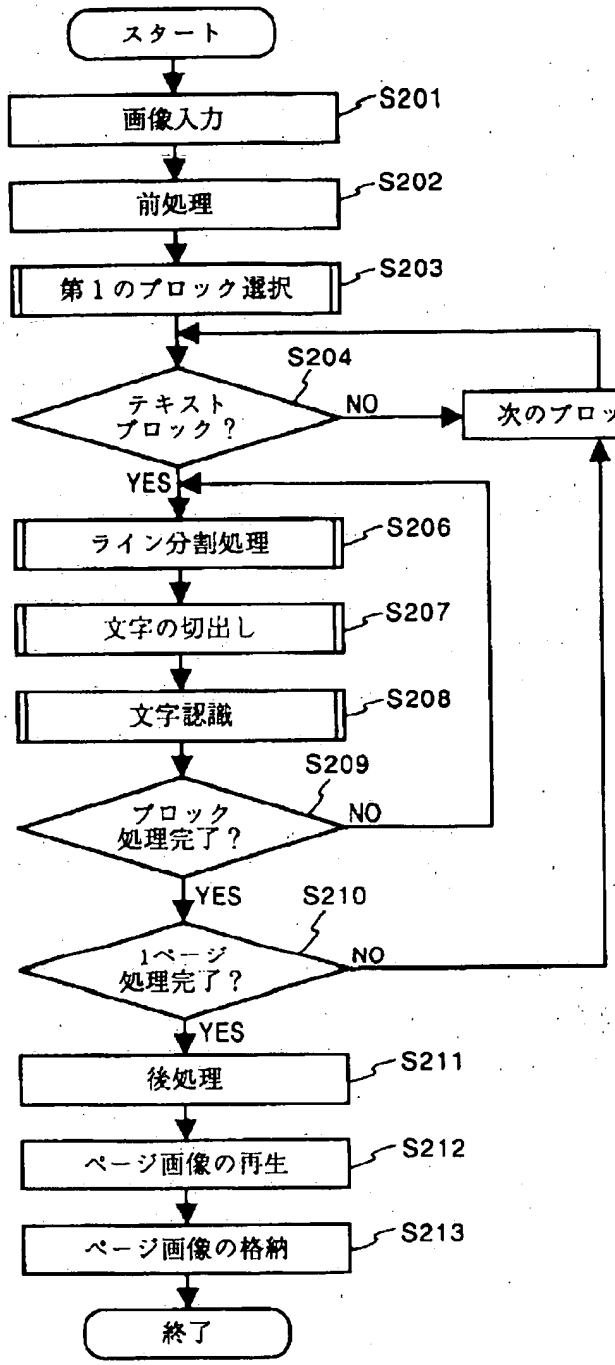
【図194】

```
freei(char **) m);
}
```

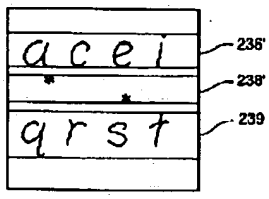
【図126】

```
#define FILE_LENGTH          80
#define LINE_LENGTH          256
#define FONT_NAME_LENGTH     20
```

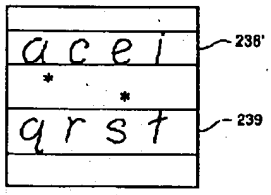
【図2】



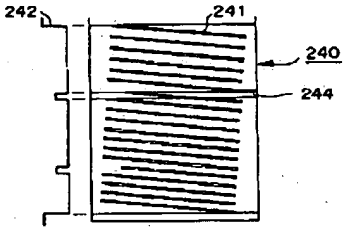
【図18C】



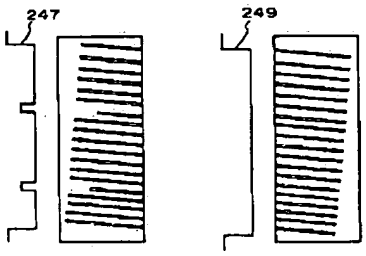
【図18D】



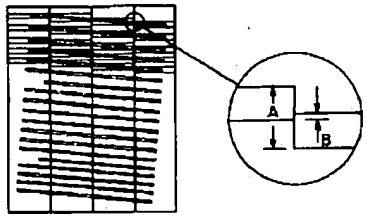
【図19A】



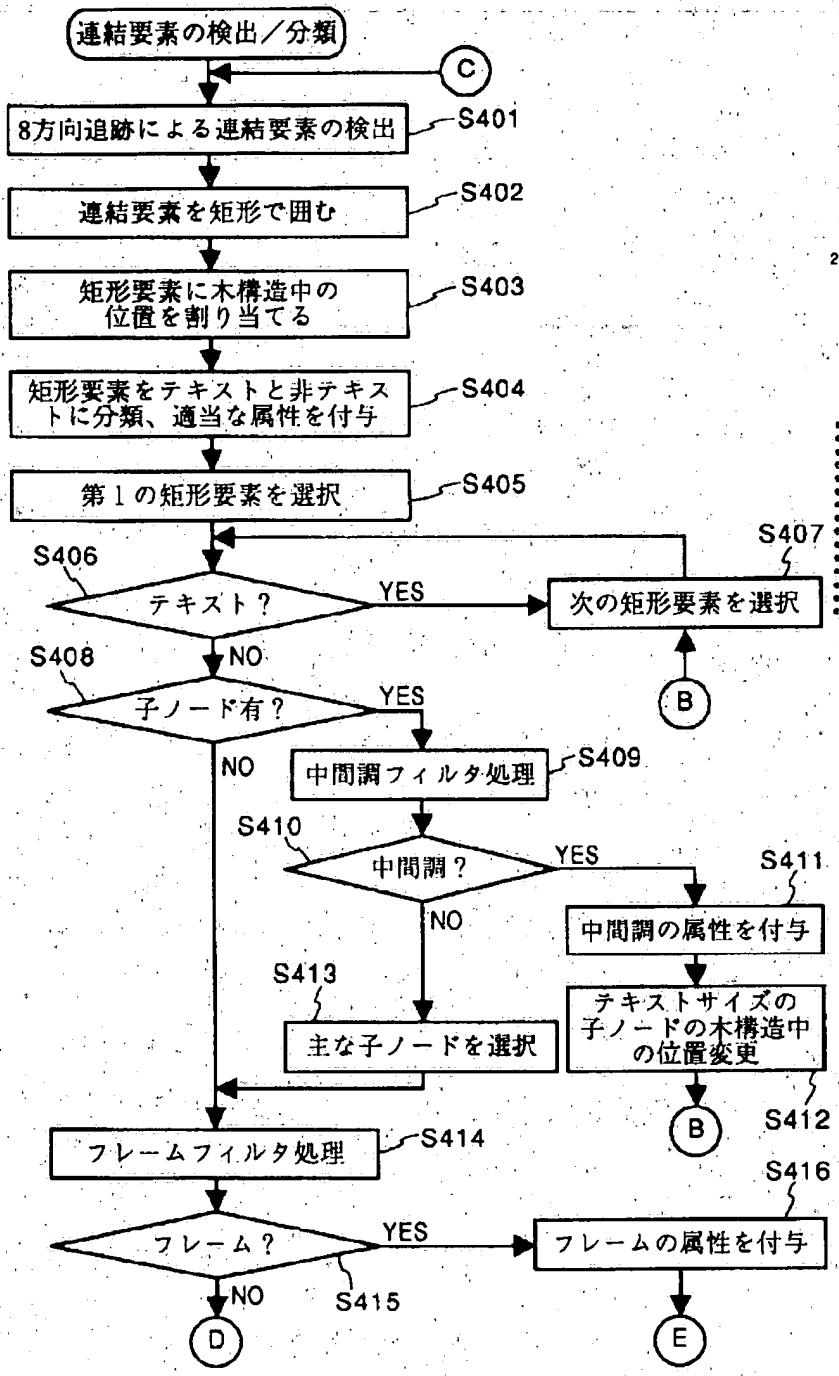
【図19B】



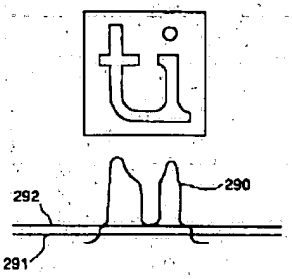
【図19D】



【図４Ａ】



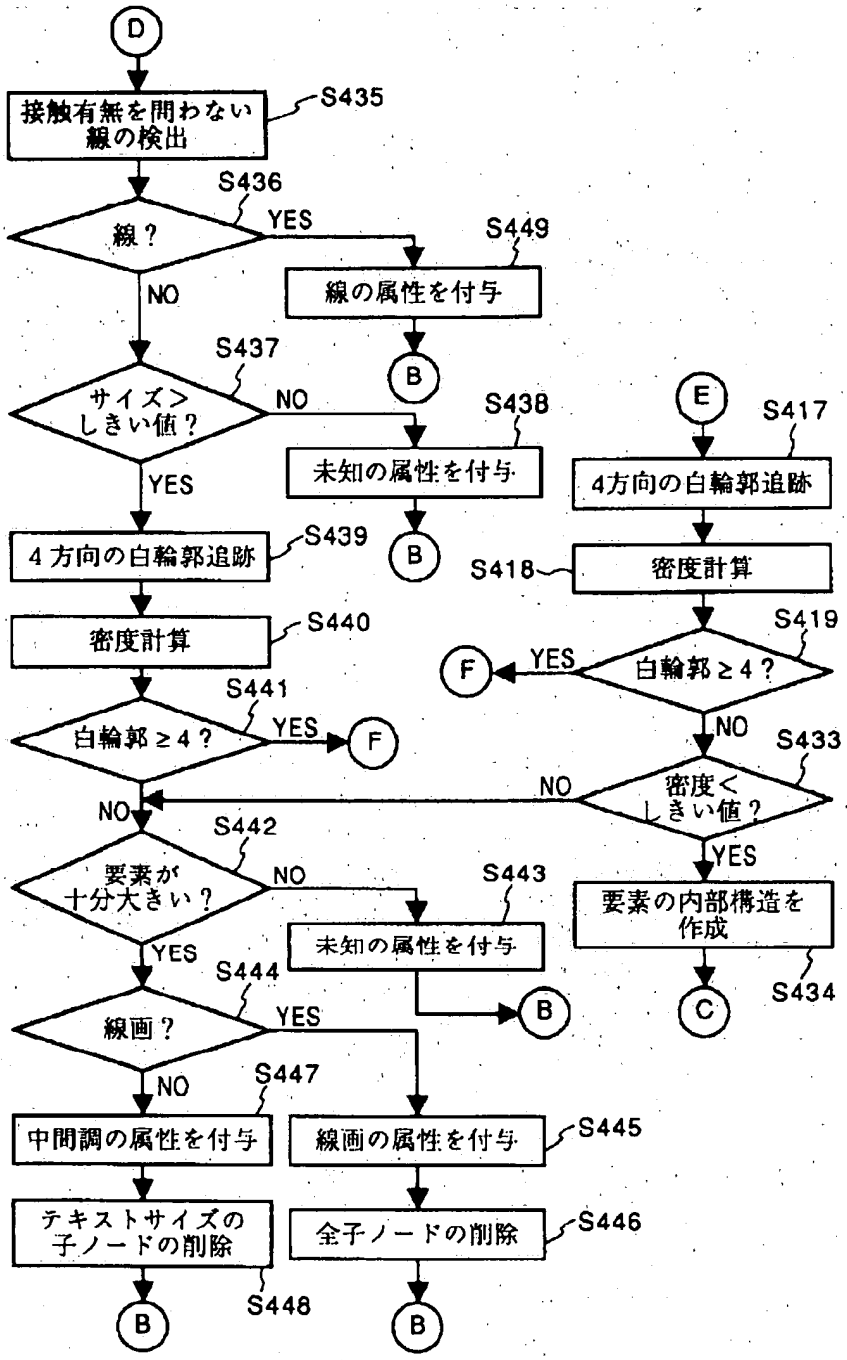
【図２９Ａ】



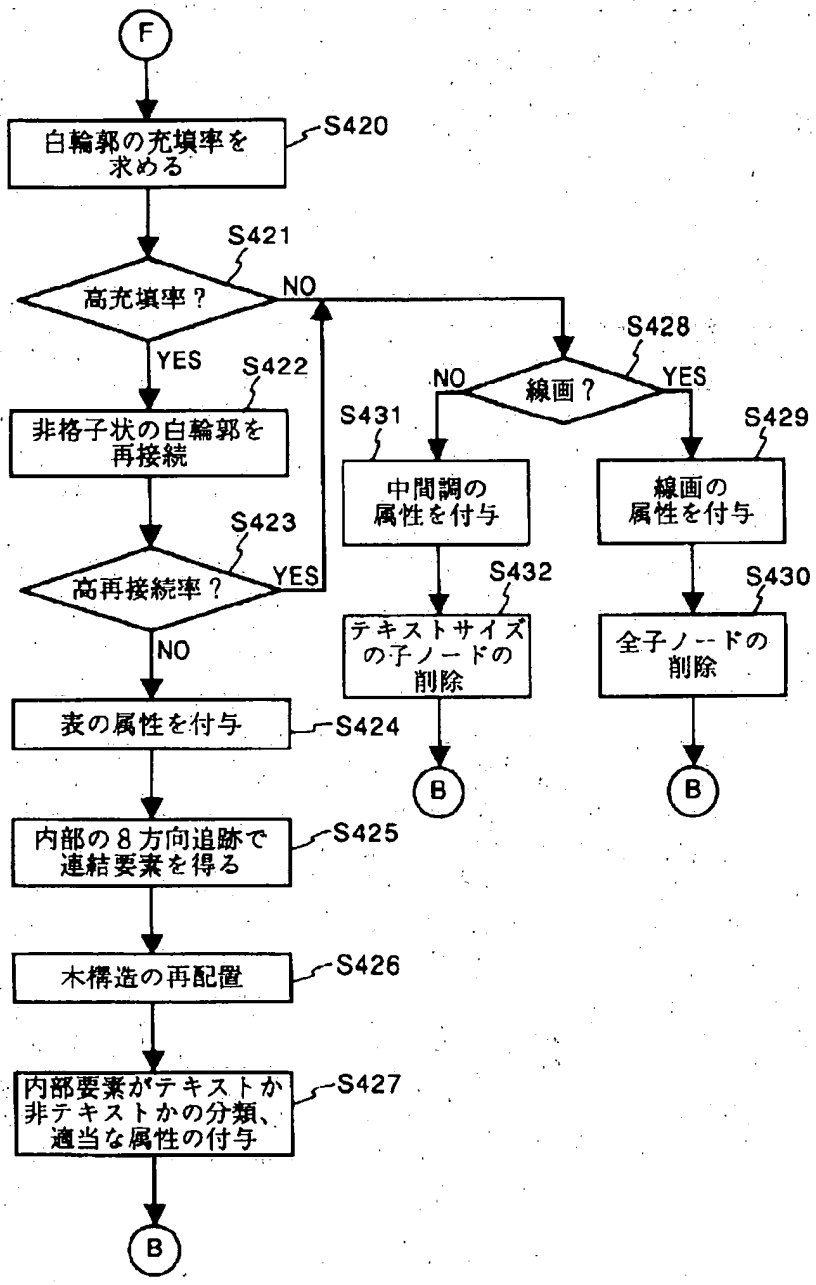
【図１３４】

```
extern   float *vector();
extern   unsigned char *cvector();
extern   int *ivector();
extern   double *dvector();
extern   unsigned char **cmatrix();
extern   float **matrix();
extern   int **imatrix();
extern   double **dmatrix();
extern   void free_vector();
extern   void free_ivector();
extern   void free_dvector();
extern   void free_matrix();
extern   void free_imatrix();
extern   void free_dmatrix();
extern   void free_cmatrix();
```

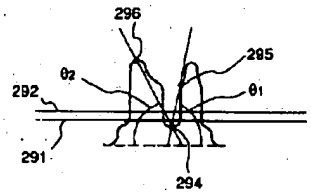
【図4B】


D
接触有無を問わない線の検出
S435
S436
線？
YES
S449
線の属性を付与
NO
B
S437
サイズ＞しきい値？
NO
S438
未知の属性を付与
YES
4方向の白輪郭追跡
S439
B
密度計算
S440
S441
白輪郭≧4？
YES
F
NO
S442
要素が十分大きい？
NO
S443
未知の属性を付与
YES
S444
線画？
YES
B
NO
S447
中間調の属性を付与
線画の属性を付与
S445
テキストサイズの子ノードの削除
全子ノードの削除
S446
S448
B
B
E
S417
4方向の白輪郭追跡
S418
密度計算
S419
F
YES
白輪郭≧4？
NO
S433
NO
密度＜しきい値？
YES
要素の内部構造を作成
S434
C

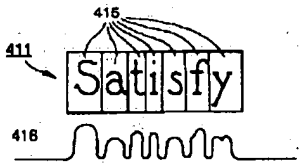
【図４Ｃ】



【図２９Ｂ】



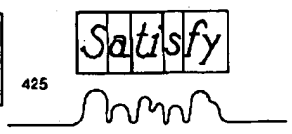
【図３４Ａ】



【図２６Ａ】



【図３４Ｂ】

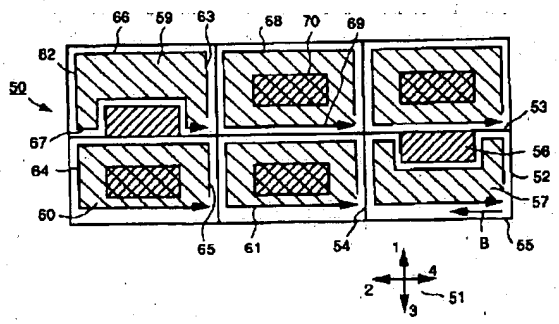


【図１６３】

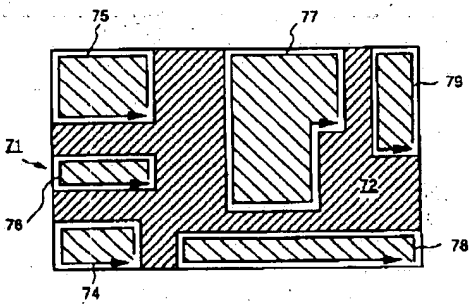
```
                                    HISTORESIS_THRESHOLD);
    charon = charon->next;          // Get the next character
}

free_ivector(histogram);            // Deallocate the memory
}
```

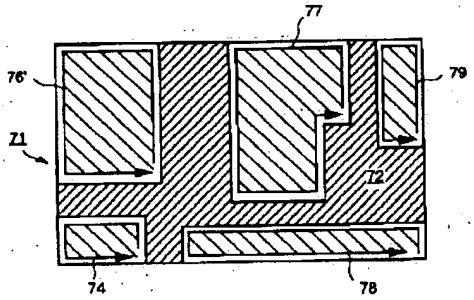
【図7A】



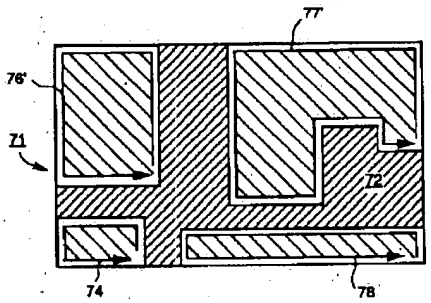
【図7B】



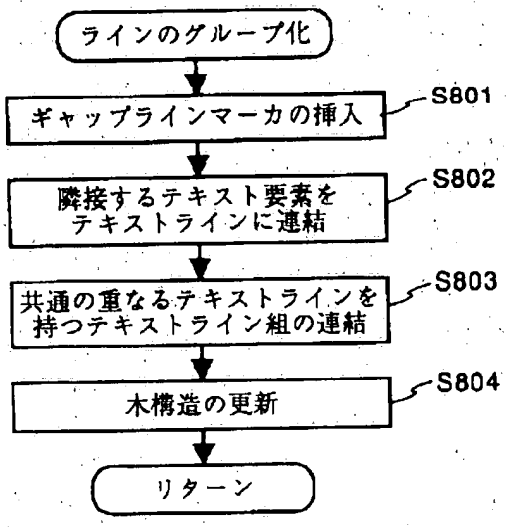
【図7C】



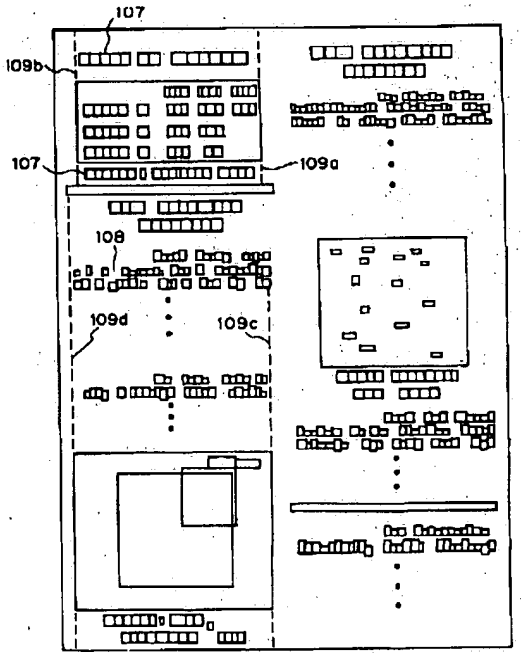
【図7D】



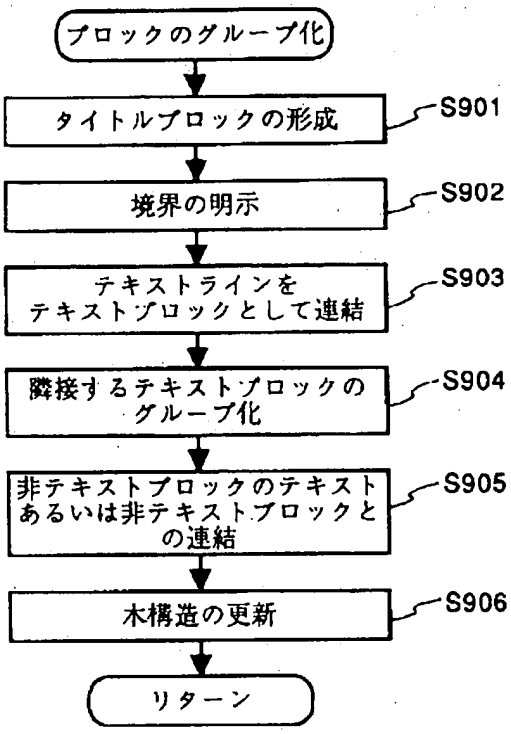
【図8】



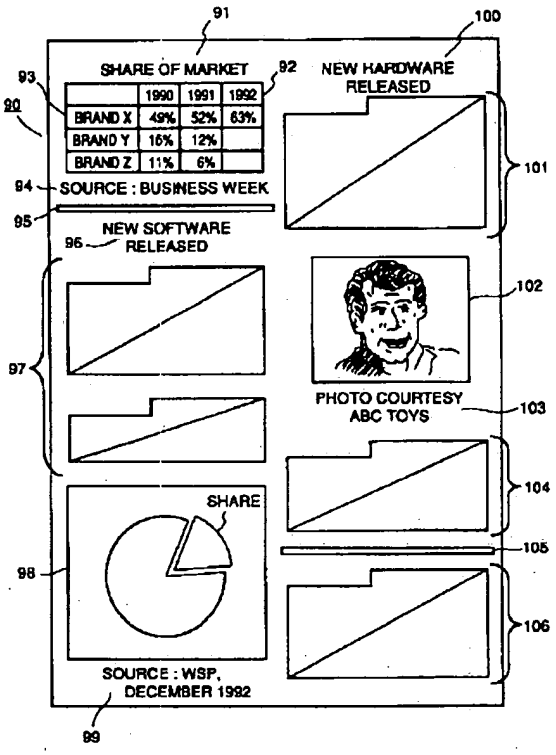
【図11】



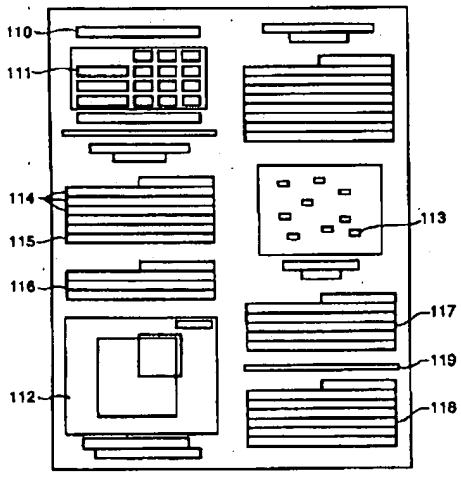
【図９】



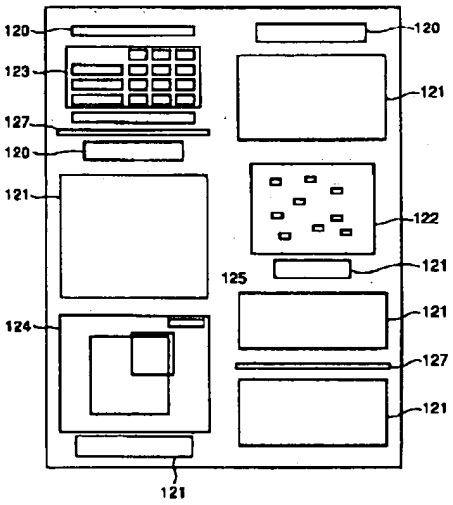
【図１０】



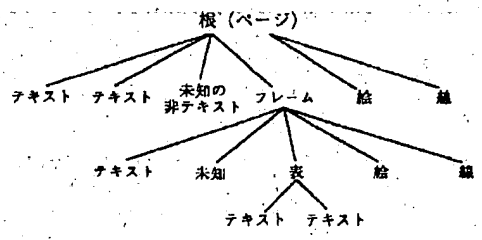
【図１２】

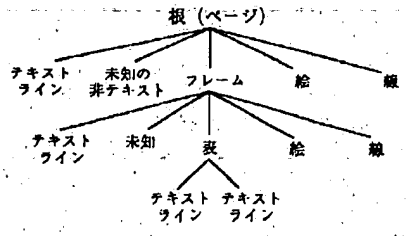


【図１３】

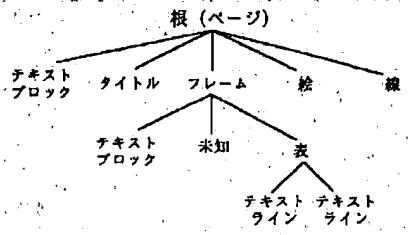


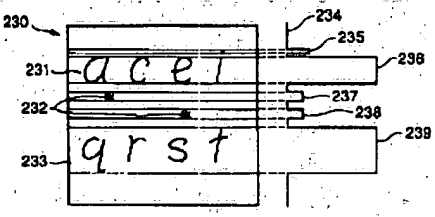
【図１４】　＊　＊　【図１５】



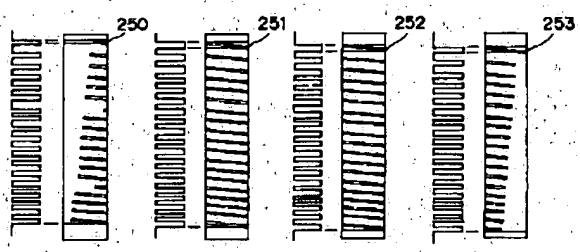


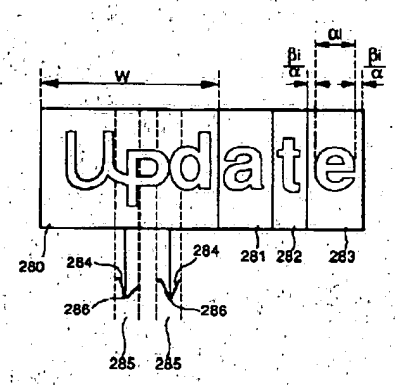
【図１６】　※　※　【図１８Ａ】



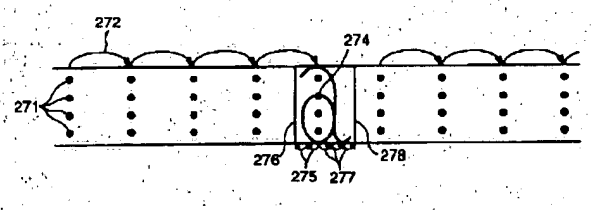


【図１９Ｃ】　★　★　【図２４】

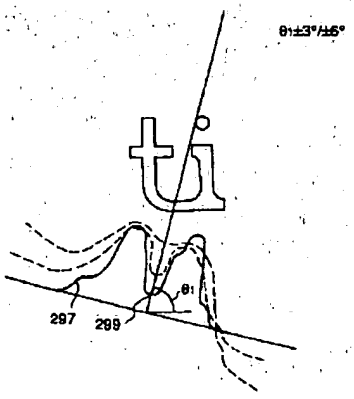




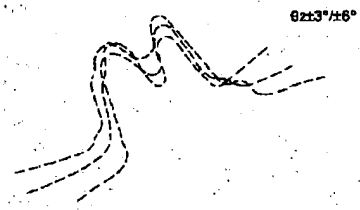
【図２２】



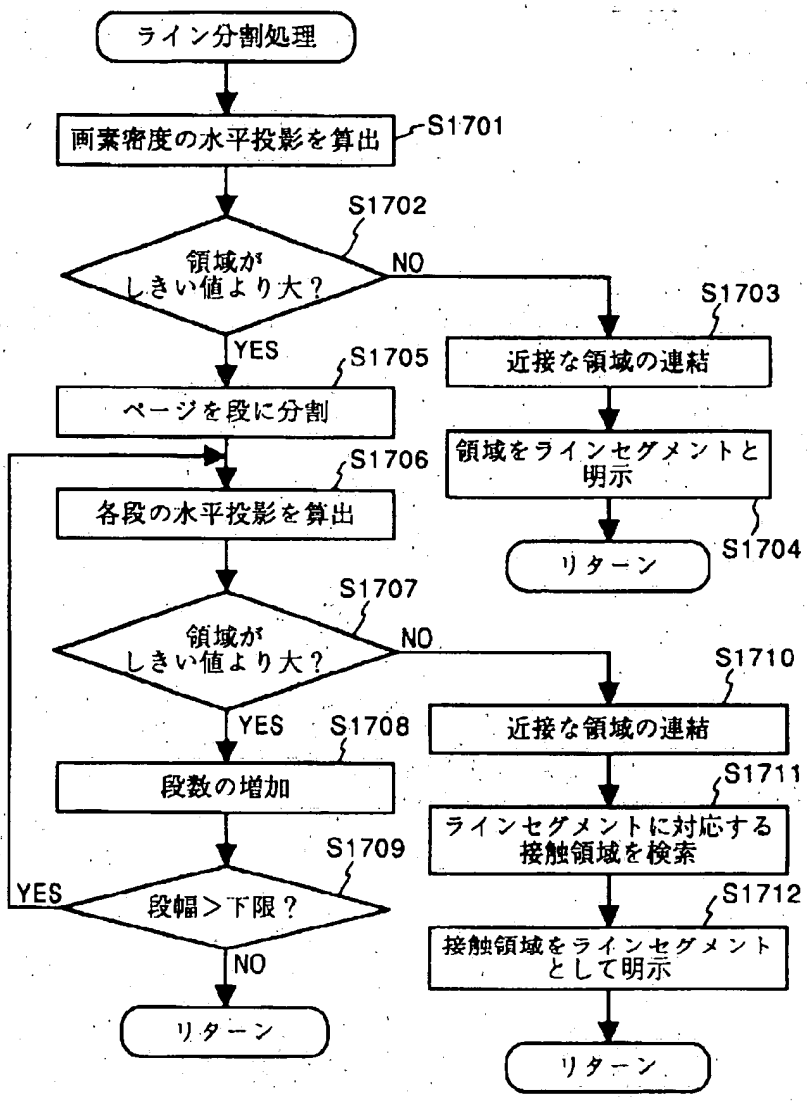
【図２９Ｃ】



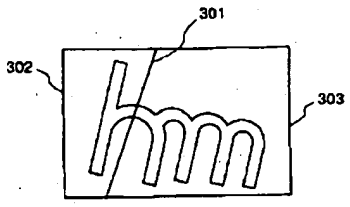
【図２９Ｄ】



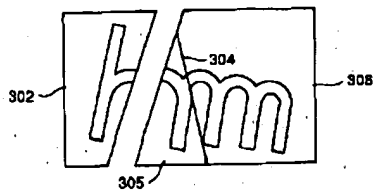
【図17】



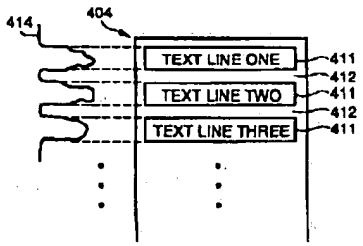
【図31A】



【図31B】



【図33A】



【図20】

【図36】

```
#include "bxlib.h"

main(argc,argv)
unsigned int argc;
char  **argv;
{
    xvidget sw, popup_Block5el();

    /*-------------------------------------------------------------*/
    /*    Initialise Architect    */
    /*-------------------------------------------------------------*/

    DxTopLevel = DxInitialize("OCR Block Selection Demo", &argc, argv);

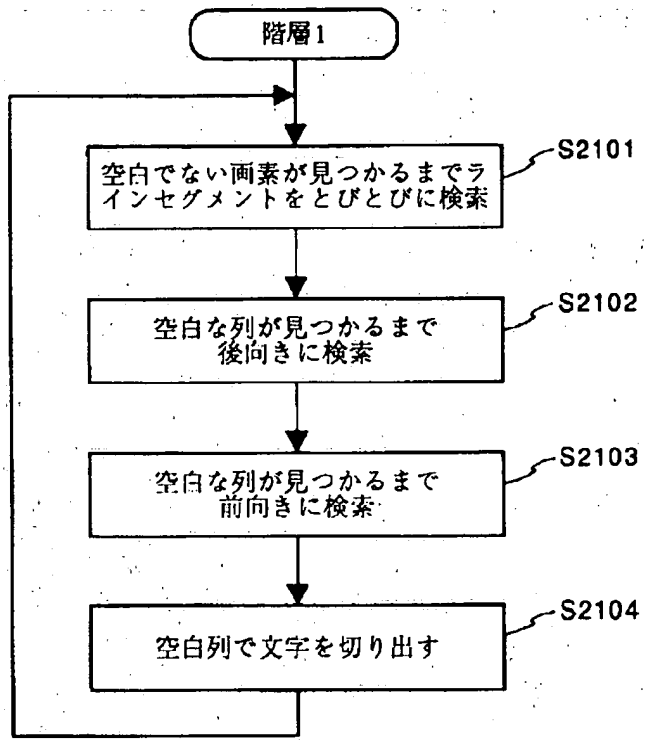
    /*-------------------------------------------------------------*/
    /* Create and popup the first window of the interface.         */
    /*-------------------------------------------------------------*/

    sw = popup_Block5el();

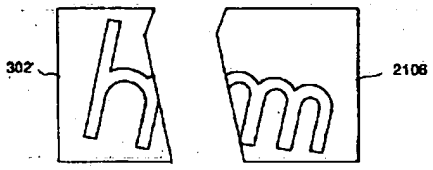
    /*-------------------------------------------------------------*/
    /* Enter the event loop; this function never returns.  */
    /*-------------------------------------------------------------*/

    DxMainLoop();
}
```

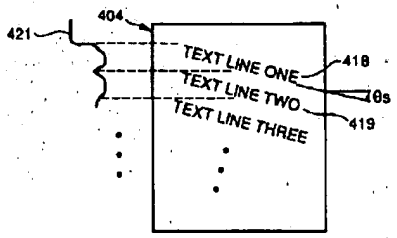
【図２１】



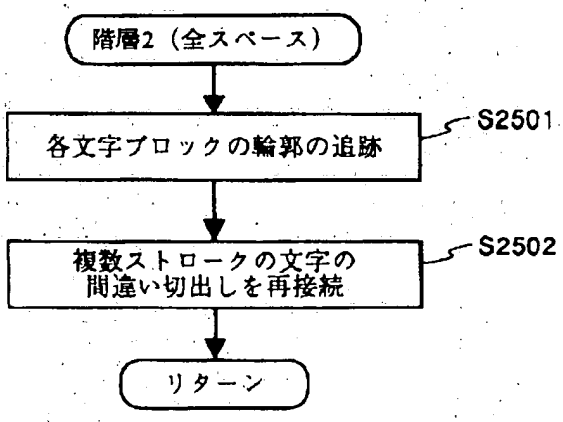
【図３１Ｃ】



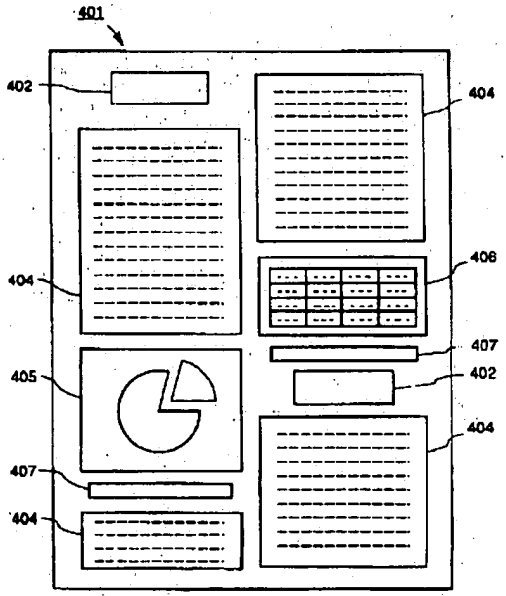
【図３３Ｂ】



【図２５】



【図３２】

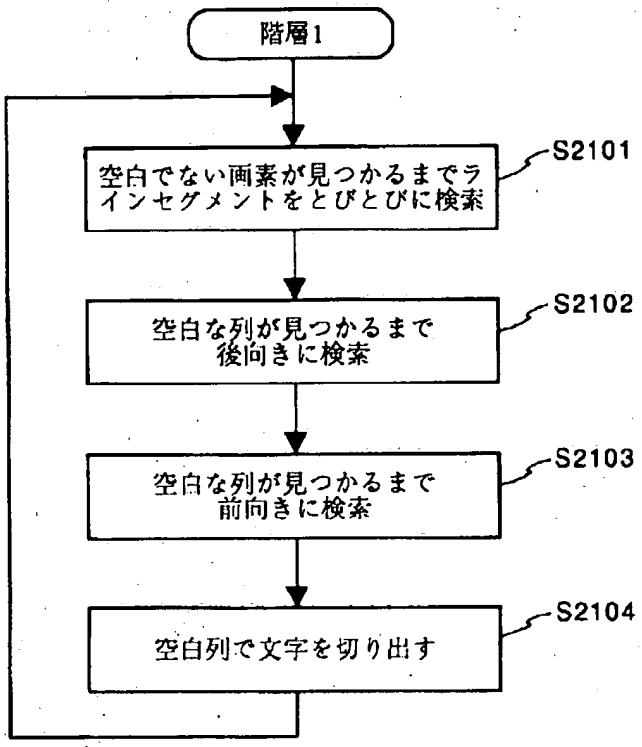


【図１２９】

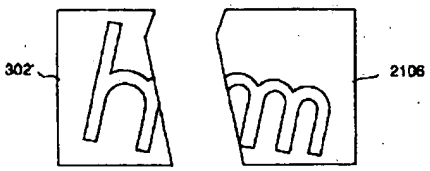
```
struct sectionblock *previous;
struct sectionblock *next;

int     upper_y, lower_y, left_x, right_x;
int     width, length;
int     index;
int     numlineblock;
int     att;
int     boundary_index;
};
```

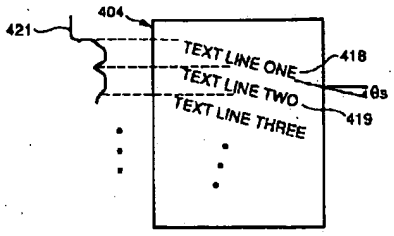
【図21】



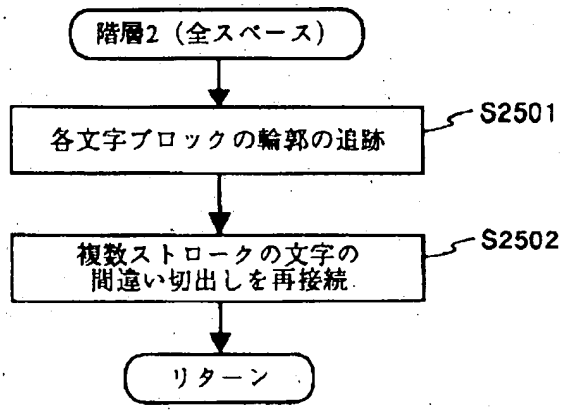
【図31C】



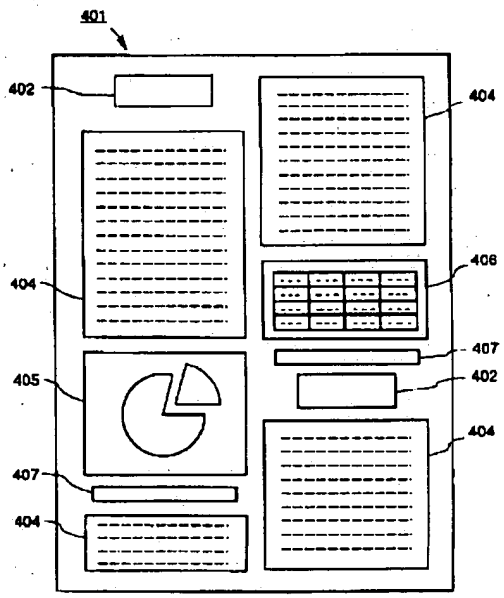
【図33B】



【図25】



【図32】

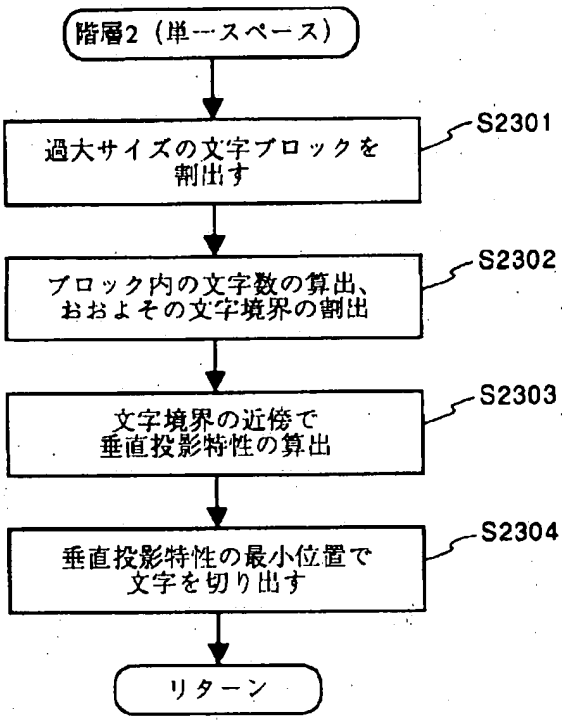


【図129】

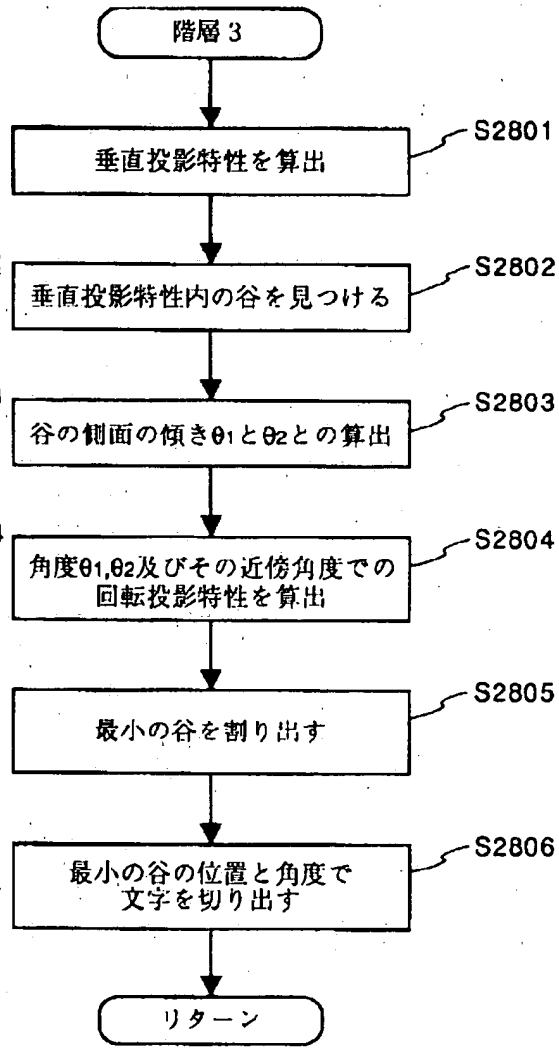
```
struct sectionblock *previous;
struct sectionblock *next;

int     upper_y, lower_y, left_x, right_x;
int     width, length;
int     index;
int     multiblock;
int     att;
int     boundary_index;
};
```

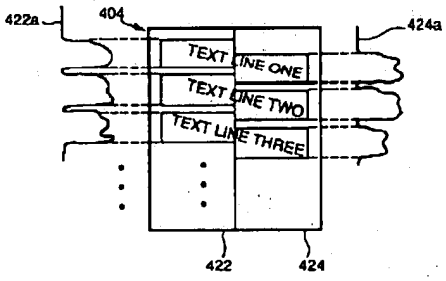
【図２３】



【図２８】



【図３３Ｃ】



【図４４】

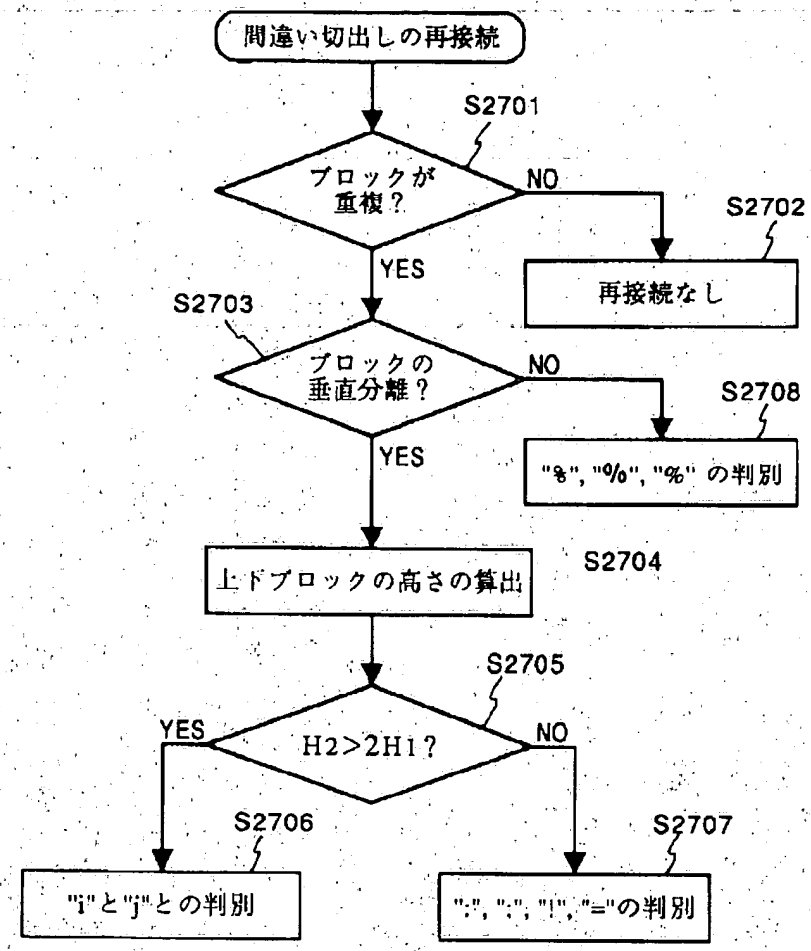
```
/* ARGSUSED */
static void cbCallback_QuitMessageBox(UxWidget, UxContext, UxCallbackArg)
        Widget                  UxWidget;
        _UxCQuitDialog          *UxContext;
        int                     UxCallbackArg;
{
        swidget                 UxThisWidget;

        UxThisWidget = UxWidgetToSwidget(UxWidget);
        _UxFromContext(UxContext);

        {
          exit();
        }

        _UxToContext(UxContext);
}
```

【図２７】



【図４６】

```
        _UxCon->DisplayForm= DisplayForm;
        _UxCon->DisplayFormClose_PBG= DisplayFormClose_PBG;
        _UxCon->DisplayForm_DrawArea= DisplayForm_DrawArea;
}

static void _UxFromContext(_UxCon)
        _UxCDisplayForm *_UxCon;
{
        DisplayForm= _UxCon->DisplayForm;
        DisplayFormClose_PBG= _UxCon->DisplayFormClose_PBG;
        DisplayForm_DrawArea= _UxCon->DisplayForm_DrawArea;
}

/* ARGSUSED */
static void activateCallback_DisplayFormClose_PBG(UxWidget, UxContext, UxCallbackArg)
        Widget          UxWidget;
        _UxCDisplayForm *UxContext;
        int             UxCallbackArg;
{
        swidget         UxThisWidget;

        UxThisWidget= UxWidgetToSwidget(UxWidget);
        _UxFromContext(UxContext);

        {
            UxPopdownInterface (DisplayForm);
        }

        _UxToContext(UxContext);
}
```

【図１３３】

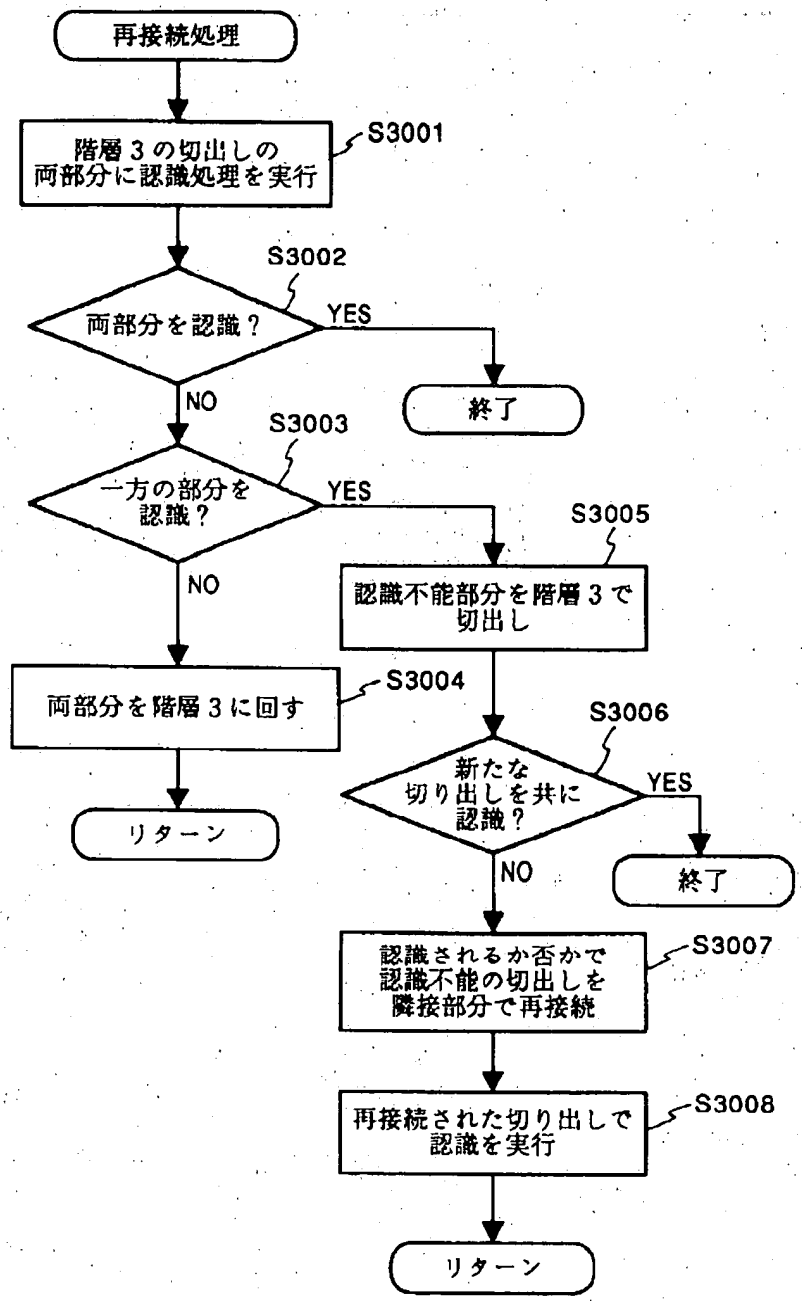
```
#define R               0
#define G               1
#define B               2

int color_comp[9][3]={{0x65, 0xcb, 0x00},
                      {0xfe, 0xf0, 0x00},
                      {0xfe, 0x00, 0xfe},
                      {0xfe, 0xfe, 0x00},
                      {0xfe, 0x98, 0x00},
                      {0x98, 0x00, 0x00},
                      {0x65, 0xfe, 0xfe},
                      {0x65, 0x00, 0xfe},
                      {0x4b, 0x4e, 0x5a}};

/*
int color_comp[9][3]={{0x00, 0xC0, 0x00},
                      {0x00, 0xC0, 0x00},
                      {0x00, 0xC0, 0x00},
                      {0x00, 0x00, 0x00},
                      {0x00, 0x00, 0x00},
                      {0x00, 0x00, 0x00},
                      {0x00, 0x00, 0x00},
                      {0x00, 0x00, 0x00},
                      {0x00, 0x00, 0x00}};

*/
```

【図30】


再接続処理
階層3の切出しの両部分に認識処理を実行
S3001
S3002
両部分を認識？
YES
終了
NO
S3003
一方の部分を認識？
YES
S3005
認識不能部分を階層3で切出し
NO
両部分を階層3に回す
S3004
リターン
S3006
新たな切り出しを共に認識？
YES
終了
NO
認識されるか否かで認識不能の切出しを隣接部分で再接続
S3007
再接続された切り出しで認識を実行
S3008
リターン

【図３７】

```
/*****************************************************************/
/*                                                               */
/*          This file was generated by Architect                 */
/*                                                               */
/*          You are free to modify or replace this header.       */
/*                                                               */
/*****************************************************************/
/*      Creation: Fri Mar 27 14:11:52 EST 1992                   */
/*****************************************************************/
#include <stdio.h>
#include "UxLib.h"
#include "UxCascB.h"
#include "UxPushBG.h"
#include "UxRowCol.h"
#include "UxLabelG.h"
#include "UxTextF.h"
#include "UxMainW.h"
#include "UxApplSh.h"
#include "UxScrWin.h"
#include "UxTextFie.h"
#include "UxFileSelectionBox.h"
extern swidget FileSelectionDialog;
extern swidget Quitdialog;
extern swidget DisplayForm;
extern swidget InfoForm;
extern swidget create_QuitDialog ();
               create_FileSelectionDialog ();
               create_DisplayForm ();
               create_InfoForm ();

typedef struct
    swidget BlockSel,
            mainWindow,
            BlockSelBB,
            menuBar,
            pullDownMenu,
            fileFane,
            scanButton,
            fileFane_b1,
            openButton,
            fileFane_b2,
            viewButton,
            fileFane_b3,
            exitButton,
            runFane,
            RunButton,
            helpFane,
            onColorRef,
            fileCascade,
            runCascade,
            helpCascade;
} _UxCBlockSel;

static void     _UxInContext(), _UxFreeContext();

static void     activateCallback_scanButton();
static void     activateCallback_openButton();
static void     activateCallback_viewButton();
static void     activateCallback_exitButton();
static void     activateCallback_RunButton();
static void     activateCallback_onColorRef();

static swidget  build_BlockSel();
static swidget  _UxpopupF_BlockSel();
static void     init_BlockSel();

swidget
static swidget
static swidget
static swidget
static swidget
static swidget
static swidget
static swidget
static swidget
static swidget
static swidget
static swidget
static swidget
static swidget
static swidget
static swidget

extern void block_process();
extern void color_reference();
extern swidget InfoFormMrm;
extern swidget DisplayForm_Drawarea;

swidget popup_BlockSel()
{
    swidget        _Uxrtrn;
    _Uxrtrn = _UxpopupF_BlockSel();

    if(UxIsValidWidget(BlockSel))
        _UxInContext(_UxCBlockSel *) UxGetContext(BlockSel));
    return(_Uxrtrn);
}

static swidget _UxpopupF_BlockSel()
{
    swidget rtrn;

    rtrn = build_BlockSel();

    create_QuitDialog (rtrn);
    create_FileSelectionDialog (rtrn);
        create_DisplayForm ();
        create_InfoForm ();
        UxPopupInterface (rtrn, no_grab);
        return(rtrn);
}

static swidget build_BlockSel()
{
    _UxCBlockSel    *_UxCon;

    _UxCon = (_UxCBlockSel *) UxAlloc(sizeof(_UxCBlockSel));

    BlockSel = UxCreateApplicationShell('BlockSel', NO_PARENT);
    UxPutContext(BlockSel, _UxCon);
    mainWindow = UxCreateMainWindow('mainWindow', BlockSel);
    BlockSelBB = UxCreateBulletinBoard('BlockSelBB', mainWindow);
    MENUBAR = UxCreateMenuBar('menuBar', mainWindow);
    pullDownMenu = UxCreateRowColumn('pullDownMenu', mainWindow);
    fileFane = UxCreateRowColumn('fileFane', pullDownMenu);
```

【図38】

```
scanButton: UxCreatePushButtonGadget('scanButton', filePane),
fileFane_b1: UxCreateSeparatorGadget('fileFane_b1', fileFane),
openButton: UxCreatePushButtonGadget('openButton', fileFane),
fileFane_b2: UxCreateSeparatorGadget('fileFane_b2', fileFane),
viewButton: UxCreatePushButtonGadget('viewButton', fileFane),
fileFane_b3: UxCreateSeparatorGadget('fileFane_b3', fileFane),
exitButton: UxCreatePushButtonGadget('exitButton', fileFane),
runFane UxCreateRowColumn('runFane', runFane);
RunButton= UxCreatePushButtonGadget('RunButton', runFane);
helpFane= UxCreateRowColumn('helpFane', pullDownMenu);
onColorRef= UxCreatePushButtonGadget('onColorRef', helpFane);
fileCascade= UxCreateCascadeButton("fileCascade", pullDownMenu);
runCascade= UxCreateCascadeButton("runCascade", pullDownMenu);
helpCascade= UxCreateCascadeButton("helpCascade", pullDownMenu);

_UxToContext(_UxCon);

int_BlockSel(BlockSel);

UxCreateWidget(mainWindow);

UxCreateWidget(BlockSel);

UxCreateWidget(README);

UxCreateWidget(pullDownMenu);

UxCreateWidget(fileFane);

UxCreateWidget(scanButton, "activateCallback", activateCallback_scanButton, _U
UxAddCallback(scanButton, "activateCallback", activateCallback_scanButton, _U
    xCon);

UxCreateWidget(openButton);
UxAddCallback(openButton, "activateCallback", activateCallback_openButton, _U
    xCon);

UxCreateWidget(fileFane_b2);

UxCreateWidget(viewButton);
UxAddCallback(viewButton, "activateCallback", activateCallback_viewButton, _U
    xCon);

UxCreateWidget(fileFane_b3);

UxCreateWidget(exitButton);
UxAddCallback(exitButton, "activateCallback", activateCallback_exitButton, _U
    xCon);

UxCreateWidget(runFane);

UxCreateWidget(RunButton);
UxAddCallback(RunButton, "activateCallback", activateCallback_RunButton, _UxC
    on);

UxCreateWidget(helpFane);

UxCreateWidget(onColorRef);
UxAddCallback(onColorRef, "activateCallback", activateCallback_onColorRef, _U
    xCon);

UxPutSubMenuId(fileCascade, "fileFane");
UxCreateWidget(fileCascade);

UxPutSubMenuId(runCascade, "runFane");
UxCreateWidget(runCascade);

UxPutSubMenuId(helpCascade, "helpFane");
UxCreateWidget(helpCascade);

UxPutMenuHelpWidget(pullDownMenu, "helpCascade");

UxRealizeInterface(BlockSel);
UxMainWindowSetAreas(mainWindow, pullDownMenu, NULL_SWIDGET,
                     NULL_SWIDGET, NULL_SWIDGET, BlockSelBB);

return(BlockSel);
}

static void init_BlockSel(_Uxw)
swidget     _Uxw;
{
    _UxFromContext( (_UxCBlockSel *) UxGetContext(_Uxw));

    UxPutX(BlockSel, (int)230);
    UxPutY(BlockSel, (int)10);
    UxPutWidth(BlockSel, (int)406);
    UxPutHeight(BlockSel, (int)180);
    UxPutTitle(BlockSel, "OCR Block selection Demo");

    UxPutX(mainWindow, (int)200);
    UxPutY(mainWindow, (int)80);
    UxPutWidth(mainWindow, (int)801);
    UxPutHeight(mainWindow, (int)100);

    UxPutWidth(BlockSelBB, (int)400);
    UxPutHeight(BlockSelBB, (int)50);

    UxPutX(README, (int)7);
    UxPutY(README, (int)21);
    UxPutWidth(README, (int)124);
    UxPutHeight(README, (int)106);
    UxPutLabelString(README, "Block Selection Demo!\n\nCanon Information Systems
\nCopyright 1992 -- Version 0.1\nMing-Yuan Wang\n");

    UxPutBorderWidth(pullDownMenu, (int)0);
    UxPutRowColumnType(pullDownMenu, "menu_bar");

    UxPutRowColumnType(fileFane, "menu_pulldown");

    UxPutLabelString(scanButton, "Scan ...");
    UxPutMnemonic(scanButton, 'S');

    UxPutLabelString(openButton, "Open ...");
    UxPutMnemonic(openButton, 'O');

    UxPutLabelString(viewButton, "View ...");
    UxPutMnemonic(viewButton, 'V');

    UxPutLabelString(exitButton, "Exit");
    UxPutMnemonic(exitButton, 'E');

    UxPutRowColumnType(runFane, "menu_pulldown");

    UxPutLabelString(RunButton, "Begin block selection process ...");
    UxPutMnemonic(RunButton, 'B');

    UxPutRowColumnType(helpFane, "menu_pulldown");

    UxPutLabelString(onColorRef, "Color Reference...");
```

【図３９】

```
    UxPutMnemonic(onColorRef, 'C');

    UxPutLabelString(fileCascade, 'File');
    UxPutMnemonic(fileCascade, 'F');

    UxPutLabelString(runCascade, 'Run');
    UxPutMnemonic(runCascade, 'R');

    UxPutLabelString(helpCascade, 'Help');
    UxPutMnemonic(helpCascade, 'H');

    _UxOContext((_UxCBlockSel *) UxGetContext(_Uxsw));
}

static void _UxCreateCon(_UxCon)
    _UxCBlockSel    *_UxCon;
{
    _UxCon->BlockSel= BlockSel;
    _UxCon->mainWindow= mainWindow;
    _UxCon->BlockSelBB= BlockSelBB;
    _UxCon->README= README;
    _UxCon->pullDownMenu= pullDownMenu;
    _UxCon->filePane= filePane;
    _UxCon->scanButton= scanButton;
    _UxCon->filePane_b1= filePane_b1;
    _UxCon->openButton= openButton;
    _UxCon->filePane_b2= filePane_b2;
    _UxCon->viewButton= viewButton;
    _UxCon->filePane_b3= filePane_b3;
    _UxCon->editButton= editButton;
    _UxCon->runPane= runPane;
    _UxCon->RunButton= RunButton;
    _UxCon->helpPane= helpPane;
    _UxCon->onColorRef= onColorRef;
    _UxCon->fileCascade= fileCascade;
    _UxCon->runCascade= runCascade;
    _UxCon->helpCascade= helpCascade;
}

static void _UxFreeContext(_UxCon)
    _UxCBlockSel    *_UxCon;
{
    BlockSel= _UxCon->BlockSel;
    mainWindow= _UxCon->mainWindow;
    BlockSelBB= _UxCon->BlockSelBB;
    README= _UxCon->README;
    pullDownMenu= _UxCon->pullDownMenu;
    filePane= _UxCon->filePane;
    scanButton= _UxCon->scanButton;
    filePane_b1= _UxCon->filePane_b1;
    openButton= _UxCon->openButton;
    filePane_b2= _UxCon->filePane_b2;
    viewButton= _UxCon->viewButton;
    filePane_b3= _UxCon->filePane_b3;
    exitButton= _UxCon->editButton;
    runPane= _UxCon->runPane;
    RunButton= _UxCon->RunButton;
    helpPane= _UxCon->helpPane;
    onColorRef= _UxCon->onColorRef;
    fileCascade= _UxCon->fileCascade;
    runCascade= _UxCon->runCascade;
    helpCascade= _UxCon->helpCascade;
}

/* ARGSUSED */
static void activateCallback_scanButton(UxWidget, UxContext, UxCallbackArg)
        Widget          UxWidget;
        XtPointer       UxContext;
        XtPointer       UxCallbackArg;
{
    _UxCBlockSel    *UxContext;
    int             UxCallbackArg;
{
    swidget         UxThisWidget;

    UxThisWidget= UxWidgetToSwidget(UxWidget);
    _UxFromContext(UxContext);

    {
        handle scanner_handle;
        scanner_handle = UxCreateSubproc ("/usr/bin/terrascan", NULL, UxAppendTo)

        if (scanner_handle == -1) {
            error_create_subproc ();
            return;
        }
        if (UxRunSubproc (scanner_handle, NULL) == -1) {
            error_run_subproc ();
            return;
        }
    }

    _UxToContext(UxContext);
}

/* ARGSUSED */
static void activateCallback_openButton(UxWidget, UxContext, UxCallbackArg)
        Widget          UxWidget;
        XtPointer       UxContext;
        int             UxCallbackArg;
{
    swidget         UxThisWidget;

    UxThisWidget= UxWidgetToSwidget(UxWidget);
    _UxFromContext(UxContext);

    {
    /*
        UxPutTitle(fileSelectionDialog, "Block Selection Java -- Open");
        UxPutSelectionLabelString(FileSelectionBox, "File to Open");
    */
        UxPopupInterface(FileSelectionDialog, no_grab);
    }

    _UxToContext(UxContext);
}

/* ARGSUSED */
static void activateCallback_viewButton(UxWidget, UxContext, UxCallbackArg)
        Widget          UxWidget;
        XtPointer       UxContext;
        _UxCBlockSel    *UxContext;
        int             UxCallbackArg;
{
    swidget         UxThisWidget;

    UxThisWidget= UxWidgetToSwidget(UxWidget);
    _UxFromContext(UxContext);

    {
        handle view_handle;
        view_handle = UxCreateSubproc ("/usr/local/bin/X11/imageview", NULL, UxAp
pendTo);
        if (view_handle == -1) {
            error_create_subproc ();
            return;
        }
        if (UxRunSubproc (view_handle, NULL) == -1) {
```

【図４０】

```
        error_fwr_subproc ();
        return;
    }
}

    _UxToContext(UxContext);
}

/* ARGSUSED */
static void activateCallback_exitButton(UxWidget, UxContext, UxCallbackArg)
    Widget  UxWidget;
    XtPointer   *UxContext;
    int     UxCallbackArg;
{
    swidget     UxThisWidget;

    UxThisWidget = UxWidgetToSwidget(UxWidget);
    _UxFromContext(UxContext);

    {
        UxPopupInterface(QuitDialog, no_grab);
    }

    _UxToContext(UxContext);
}

/* ARGSUSED */
static void activateCallback_RunButton(UxWidget, UxContext, UxCallbackArg)
    Widget  UxWidget;
    XtPointer   *UxContext;
    int     UxCallbackArg;
{
    swidget     UxThisWidget;
    char        *foreground, *background;
    int         height, width;

    UxThisWidget = UxWidgetToSwidget(UxWidget);
    _UxFromContext(UxContext);

    {
        /* Start execution of your program here */
        UxPopupInterface(DisplayForm, no_grab);

        height = UxGetHeight (DisplayForm_DrawArea);
/*      printf ("height => %d\n", height);
*/
        width = UxGetWidth(DisplayForm_DrawArea);
/*      printf ("width => %d\n", width);
*/

        foreground = UxGetForeground(DisplayForm_DrawArea);
        background = UxGetBackground(DisplayForm_DrawArea);
/*      printf("f = %s b = %s \n", foreground, background);
*/
        block_process(UxGetWidget(DisplayForm_DrawArea), width, height);
    }

    _UxToContext(UxContext);
}

/* ARGSUSED */
static void activateCallback_colorRef(UxWidget, UxContext, UxCallbackArg)
    Widget  UxWidget;
    XtPointer   *UxContext;
    int     UxCallbackArg;
{
    swidget     UxThisWidget;
    int         width, height;
    char        *foreground, *background;
    char        *font;

    UxThisWidget = UxWidgetToSwidget(UxWidget);
    _UxFromContext(UxContext);

    {
        UxPopupInterface(Infoform, no_grab);
        height = UxGetHeight(InfoFormDArea);
        printf ("height => %d\n", height);
        width = UxGetWidth(InfoFormDArea);
        printf ("width => %d\n", width);

        foreground = UxGetForeground(InfoFormDArea);
        background = UxGetBackground(InfoFormDArea);
/*      printf("f = %s b = %s \n", foreground, background);
*/
        font = UxGetFontList(InfoFormDArea);
/*      printf("font = %s \n", font);
*/
        color_reference(UxGetWidget(InfoFormDArea), width, height);
    }

    _UxToContext(UxContext);
}
```

【図41】

```
/*****************************************************************/
/****                                                         ****/
/****          This file was generated by Architect           ****/
/****                                                         ****/
/****       You are free to modify or replace this header.    ****/
/****                                                         ****/
/*****************************************************************/
/****  ..... ****/
/****    Creation: Wed Mar 25 20:37:43 EST 1992               ****/
/*****************************************************************/
#include <stdio.h>
#include "UxLib.h"
#include "UxFBox.h"
#include "UxDialSh.h"
#include "UxSubproc.h"
#include "block.h"
static XmStringCharSet charset = (XmStringCharSet) XmSTRING_DEFAULT_CHARSET;
char filename[FILE_LENGTH];

typedef struct
        {
        swidget FileSelectionDialog,
                FileSelectionBox;
        swidget parent;
} _UxCFileSelectionDialog;

static void     _UxFromContext(), _UxFreeContext();

static void     cancelCallback_FileSelectionBox(), okCallback_FileSelectionBox();

static swidget  build_FileSelectionDialog();
static swidget  _UxCreate_FileSelectionDialog();
static void     init_FileSelectionDialog();

swidget         FileSelectionDialog;
swidget         FileSelectionBox;

extern void open_tiff_file();

static swidget  parent;

char *get_input_filename ()
{
    int a = 0;
    Arg args[10];
    XmString str;
    char *name;

    XtSetArg (args[a], XmNtextString, &str); a++;
    XtGetValues (UxGetWidget(FileSelectionBox), args, a);
    XmStringGetLtoR (str, charset, &name);
    return name;
}

void OutputHandler (fd, data)
  int fd;
  char *data;
{
  printf ("In output handler\n");
}

swidget create_FileSelectionDialog(_Uxparent)
        swidget _Uxparent;
{
        swidget _UxRtrn;
        _UxCFileSelectionDialog *_UxCon;

        if(UxIsValidWidget(FileSelectionDialog))
                _UxCon=(_UxCFileSelectionDialog *) UxGetContext(FileSelectionD
log));
        return(_UxCon);
        }

static swidget _Uxcreate_FileSelectionDialog(_Uxparent)
        swidget _Uxparent;
{
        swidget rtrn;

        parent = _Uxparent;
        rtrn = build_FileSelectionDialog();

        XtManageChild(XmSelectionBoxGetChild(UxGetWidget(FileSelectionBox), XmDIALO
G_HELP_BUTTON));
        return(rtrn);
}

static swidget build_FileSelectionDialog()
{
        _UxCFileSelectionDialog *_UxCon;

        _UxCon= (_UxCFileSelectionDialog *) UxMalloc(sizeof(_UxCFileSelectionDialog));

        _UxCon->parent= parent;

        FileSelectionDialog= UxCreateDialogShell("FileSelectionDialog", parent);
        UxPutContext(FileSelectionDialog, _UxCon);
        FileSelectionBox= UxCreateFileSelectionBox("FileSelectionBox", FileSelectionD
ialog);

        _UxToContext(_UxCon);

        init_FileSelectionDialog(FileSelectionDialog);

        UxCreateWidget(FileSelectionBox);
        UxAddCallback(FileSelectionBox, "cancelCallback", cancelCallback_FileSelectio
nBox, _UxCon);
        UxAddCallback(FileSelectionBox, "okCallback", okCallback_FileSelectionBox, _U
xCon);

        UxRealizeInterface(FileSelectionDialog);

        return(FileSelectionDialog);
}

static void init_FileSelectionDialog(_Uxsw)
        swidget _Uxsw;
{
        _UxCFromContext((_UxCFileSelectionDialog *) UxGetContext(_Uxsw));

        UxPutTitle(FileSelectionDialog, "File Selection");

        UxPutDialogStyle(FileSelectionBox, "dialog_primary_application_modal");

        _UxToContext((_UxCFileSelectionDialog *) UxGetContext(_Uxsw));
}

static void _UxFromContext(_UxCon)
        _UxCFileSelectionDialog *_UxCon;
{
```

【図42】

```
        _UxCon->FileSelectionDialog= FileSelectionDialog;
        _UxCon->FileSelectionBox= FileSelectionBox;

        _UxCon->parent= parent;
}

static void _UxFromContext(_UxCon)
        _UxCFileSelectionDialog *_UxCon;
{
        FileSelectionDialog= _UxCon->FileSelectionDialog;
        FileSelectionBox= _UxCon->FileSelectionBox;

        parent= _UxCon->parent;
}

/* ARGSUSED */
static void cancelCallback_FileSelectionBox(UxWidget, UxContext, UxCallbackArg)
        Widget          UxWidget;
        _UxCFileSelectionDialog *UxContext;
        int             UxCallbackArg;
{
        swidget         UxThisWidget;

        UxThisWidget= UxWidgetToSwidget(UxWidget);
        _UxFromContext(UxContext);

        {
            UxPopdownInterface(FileSelectionDialog);
        }

        _UxToContext(UxContext);
}

/* ARGSUSED */
static void okCallback_FileSelectionBox(UxWidget, UxContext, UxCallbackArg)
        Widget          UxWidget;
        _UxCFileSelectionDialog *UxContext;
        int             UxCallbackArg;

        swidget         UxThisWidget;

        UxThisWidget= UxWidgetToSwidget(UxWidget);
        _UxFromContext(UxContext);

        {
            handle tiff_handle;
            UxPopdownInterface(FileSelectionDialog);
            strcpy (filename, get_input_filename ());
            printf ("Open [%s]\n", filename);
            tiff_handle = UxCreateSubproc ("/usr/local/bin/X11/imageview", filename,
UxAppendTo);
            if (tiff_handle == -1) {
                error_create_subproc ();
                return;
            }
            if (UxRunSubproc (tiff_handle, NULL) == -1) {
                error_run_subproc ();
                return;
            }
            open_tiff_file(filename);
        }

        _UxToContext(UxContext);
}
```

【図48】

```
}

static void _UxFromContext(_UxCon)
        _UxCInfoForm    *_UxCon;
{
        InfoForm= _UxCon->InfoForm;
        InfoFormClose_PBG= _UxCon->InfoFormClose_PBG;
        InfoFormDAreas= _UxCon->InfoFormDAreas;
}

/* ARGSUSED */
static void activateCallback_InfoFormClose_PBG(UxWidget, UxContext, UxCallbackArg)
        Widget          UxWidget;
        _UxCInfoForm    *UxContext;
        int             UxCallbackArg;
{
        swidget         UxThisWidget;

        UxThisWidget= UxWidgetToSwidget(UxWidget);
        _UxFromContext(UxContext);

        {
            UxPopdownInterface (InfoForm);
        }

        _UxToContext(UxContext);
}
```

【図62】

```
int width, height;
{
    int x, y, xdis, ydis, i;
    static fid = 33;

    init_graphics(wd);
    xs_clr_win(wd);
    xdis = ((height-20)/(TOTALCOLOR));
    xs_get(font(wd);
    for ( i = 0; i < TOTALCOLOR; i++){
        xs_set_foreground(color_code[i], wd);
        xs_setfont(wd, fid);
        xs_string(wd, 20, (i+1)*xdis+10, color_label[i]);
    }
    xs_flush(wd);
}
```

[図43]

```
/*****************************************************************************/
/***                                                                       ***/
/***                    This file was generated by Architect               ***/
/***                                                                       ***/
/***          You are free to modify or replace this header.               ***/
/***                                                                       ***/
/*****************************************************************************/
/***      Creation: Wed Mar 25 20:37:32 EST 1992                           ***/
/*****************************************************************************/
#include <stdio.h>
#include "UxLib.h"
#include "UxMsgBox.h"
#include "UxDialSh.h"
#include <Xll/MwmUtil.h>

typedef struct
{       swidget QuitDialog,
                QuitMessageBox;
        swidget parent;
} _UxCQuitDialog;

static void     _UxToContext(), _UxFromContext();

static void     okCallback_QuitMessageBox();

static swidget  build_QuitDialog();
static swidget  _Uxcreate_QuitDialog();
static void     init_QuitDialog();

swidget         QuitDialog;
static swidget  QuitMessageBox;


static swidget  parent;


swidget create_QuitDialog(_Uxparent)
        swidget _Uxparent;
{
        swidget         _Uxrtrn;
        _Uxrtrn= _Uxcreate_QuitDialog(_Uxparent);

        if(UxIsValidSwidget(QuitDialog))
                _UxToContext((_UxCQuitDialog *) UxGetContext(QuitDialog));
        return(_Uxrtrn);
}

static swidget _Uxcreate_QuitDialog(_Uxparent)
        swidget _Uxparent;
{
        swidget rtrn;

        parent = _Uxparent;
        rtrn = build_QuitDialog();

        XtUnmanageChild (XmMessageBoxGetChild(UxGetWidget(QuitMessageBox),
                                XmDIALOG_HELP_BUTTON));
            return(rtrn);

}

static swidget build_QuitDialog()
{
        _UxCQuitDialog  *_UxCon;

        _UxCon= (_UxCQuitDialog *) UxAlloc(sizeof(_UxCQuitDialog));

        _UxCon->parent= parent;

        QuitDialog= UxCreateDialogShell("QuitDialog", parent);
        UxPutContext(QuitDialog, _UxCon);
        QuitMessageBox= UxCreateMessageBox("QuitMessageBox", QuitDialog);

        _UxToContext(_UxCon);

        init_QuitDialog(QuitDialog);

        UxCreateWidget(QuitDialog);

        UxCreateWidget(QuitMessageBox);
        UxAddCallback(QuitMessageBox, "okCallback", okCallback_QuitMessageBox, _UxCon
);

        UxDelayUpdate(QuitDialog);
        UxPutMwmDecorations(QuitDialog, (int)MWM_DECOR_BORDER | MWM_DECOR_TITLE | MWM
_DECOR_MENU);
        UxPutMwmFunctions(QuitDialog, (int)MWM_FUNC_MOVE | MWM_FUNC_CLOSE);
        UxUpdate(QuitDialog);

        UxRealizeInterface(QuitDialog);

        return(QuitDialog);
}

static void init_QuitDialog(_Uxsw)
        swidget         _Uxsw;
{
        _UxFromContext((_UxCQuitDialog *) UxGetContext(_Uxsw));

        UxPutX(QuitDialog, (int)490);
        UxPutY(QuitDialog, (int)410);
        UxPutWidth(QuitDialog, (int)170);
        UxPutHeight(QuitDialog, (int)120);

        UxPutDialogStyle(QuitMessageBox, "dialog_primary_application_modal");
        UxPutDialogTitle(QuitMessageBox, "Quit  Dialog");
        UxPutDialogType(QuitMessageBox, "dialog_question");
        UxPutMessageString(QuitMessageBox, "Quit Block Selection Demo?");

        _UxToContext((_UxCQuitDialog *) UxGetContext(_Uxsw));
}

static void _UxToContext(_UxCon)
        _UxCQuitDialog  *_UxCon;
{
        _UxCon->QuitDialog= QuitDialog;
        _UxCon->QuitMessageBox= QuitMessageBox;

        _UxCon->parent= parent;
}

static void _UxFromContext(_UxCon)
        _UxCQuitDialog  *_UxCon;
{
        QuitDialog= _UxCon->QuitDialog;
        QuitMessageBox= _UxCon->QuitMessageBox;

        parent= _UxCon->parent;
}
```

【図45】

```
/*****************************************************************************/
/***                                                                       ***/
/***          This file was generated by Architect                         ***/
/***                                                                       ***/
/***          You are free to modify or replace this header.               ***/
/***                                                                       ***/
/*****************************************************************************/
/***     Creation: Wed Mar 23 20:37:40 EST 1994                            ***/
/*****************************************************************************/
#include <stdio.h>
#include "UxLib.h"
#include "UxDrawAre.h"
#include "UxPushBG.h"
#include "UxForm.h"

typedef struct
{
    swidget DisplayForm;
            DisplayFormClose_PBG;
            DisplayForm_DrawArea;
} _UxCDisplayForm;

static void    _UxFromContext(); _UxFromContext;

static void    activateCallback_DisplayFormClose_PBG();

static swidget build_DisplayForm();
static swidget _UxCreate_DisplayForm();
static void    init_DisplayForm();

swidget    DisplayForm;
static swidget DisplayFormClose_PBG;
swidget    DisplayForm_DrawArea;


swidget create_DisplayForm()
{
    swidget        _UxRtrn;
    _UxRtrn= _UxCreate_DisplayForm();

    if(UxGetWidget(DisplayForm))
        _UxToContext((_UxCDisplayForm *) UxCatContext(DisplayForm));
    return(_UxRtrn);
}

static swidget _UxCreate_DisplayForm()
{
    swidget rtrn;

    rtrn = build_DisplayForm();

    return(rtrn);
}

static swidget build_DisplayForm()
{
    _UxCDisplayForm *_UxCon;

    _UxCon= (_UxCDisplayForm *) UxAlloc(sizeof(_UxCDisplayForm));

    DisplayForm= UxCreateForm("DisplayForm", NO_PARENT);
    UxPutContext(DisplayForm, _UxCon);
    UxPutDefaultShell(DisplayForm, "topLevelShell");
    DisplayFormClose_PBG= UxCreatePushButtonGadget("DisplayFormClose_PBG", DisplayForm);
    UxPutContext(DisplayFormClose_PBG, _UxCon);
    DisplayForm_DrawArea= UxCreateDrawingArea("DisplayForm_DrawArea", DisplayForm);
    UxPutContext(DisplayForm_DrawArea, _UxCon);

    _UxToContext(_UxCon);

    init_DisplayForm(DisplayForm);

    UxCreateWidget(DisplayForm);

    UxPutBottomAttachment(DisplayForm_DrawArea, "attach_form");
    UxPutBottomOffset(DisplayFormClose_PBG, (int)5);
    UxPutLeftAttachment(DisplayForm_DrawArea, "attach_form");
    UxPutLeftOffset(DisplayFormClose_PBG, (int)5);
    UxPutRightAttachment(DisplayFormClose_PBG, "attach_none");
    UxCreateWidget(DisplayFormClose_PBG);
    UxAddCallback(DisplayFormClose_PBG, "activateCallback", activateCallback_DisplayFormClose_PBG, _UxCon);

    UxPutBottomAttachment(DisplayForm_DrawArea, "attach_widget");
    UxPutBottomOffset(DisplayForm_DrawArea, (int)5);
    UxPutBottomWidget(DisplayForm_DrawArea, "DisplayFormClose_PBG");
    UxPutLeftAttachment(DisplayForm_DrawArea, "attach_form");
    UxPutLeftOffset(DisplayForm_DrawArea, (int)5);
    UxPutRightAttachment(DisplayForm_DrawArea, "attach_form");
    UxPutRightOffset(DisplayForm_DrawArea, (int)5);
    UxPutTopAttachment(DisplayForm_DrawArea, "attach_form");
    UxPutTopOffset(DisplayForm_DrawArea, (int)5);
    UxCreateWidget(DisplayForm_DrawArea);

    UxRealizeInterface(DisplayForm);

    return(DisplayForm);
}

static void init_DisplayForm(_Uxsw)
    swidget        _Uxsw;
{
    _UxFromContext((_UxCDisplayForm *) UxCatContext(_Uxsw));

    UxPutX(DisplayForm, (int)700);
    UxPutY(DisplayForm, (int)400);
    UxPutWidth(DisplayForm, (int)470);
    UxPutHeight(DisplayForm, (int)370);
    UxPutDialogTitle(DisplayForm, "MAGE  DISPLAY");

    UxPutX(DisplayFormClose_PBG, (int)10);
    UxPutY(DisplayFormClose_PBG, (int)260);
    UxPutWidth(DisplayFormClose_PBG, (int)80);
    UxPutHeight(DisplayFormClose_PBG, (int)30);
    UxPutLabelString(DisplayFormClose_PBG, "Close");
    UxPutRecomputeSize(DisplayFormClose_PBG, "false");

    UxPutX(DisplayForm_DrawArea, (int)10);
    UxPutY(DisplayForm_DrawArea, (int)50);
    UxPutWidth(DisplayForm_DrawArea, (int)450);
    UxPutHeight(DisplayForm_DrawArea, (int)250);
    UxPutBackground(DisplayForm_DrawArea, "white");
    UxPutForeground(DisplayForm_DrawArea, "yellow");
    UxPutResizePolicy(DisplayForm_DrawArea, "resize_none");

    _UxToContext((_UxCDisplayForm *) UxCatContext(_Uxsw));
}

static void activateCallback_DisplayFormClose_PBG(_UxwCon)
    _UxCDisplayForm *_UxCon;
```

【図47】

```
/*****************************************************************/
/***                                                           ***/
/***          This file was generated by Architect             ***/
/***                                                           ***/
/***      You are free to modify or replace this header.       ***/
/***                                                           ***/
/*****************************************************************/
/***      Creation: Fri Mar 27 16:14:57 EST 1992               ***/
/*****************************************************************/
#include <stdio.h>
#include "Xm/Xm.h"
#include "Xm/PushB.h"
#include "UxForm.h"

typedef struct
        {
        swidget InfoForm;
        swidget InfoFormClose_PBG;
        swidget InfoFormDraw;
        } _UxCInfoForm;

static void   _UxTxContext(), _UxFromContext();

static void   activateCallback_InfoFormClose_PBG();

static swidget build_InfoForm();
static swidget _Uxcreate_InfoForm();
static void   Init_InfoForm();

swidget       InfoForm;
static swidget InfoFormClose_PBG;
swidget InfoFormDraw;

swidget create_InfoForm()
{
    swidget _Uxrtrn;
    swidget _Uxrtrn= _Uxcreate_InfoForm();
    if(UxIsValidGvidget(InfoForm))
        _UxToContext((_UxCInfoForm *) UxGetContext(InfoForm));
    return(_Uxrtrn);
}

static swidget _Uxcreate_InfoForm()
{
    swidget rtrn;

    rtrn = build_InfoForm();

    return(rtrn);
}

static swidget build_InfoForm()
{
    _UxCInfoForm   *_UxCon;

    _UxCon= (_UxCInfoForm *) UxAlloc(sizeof(_UxCInfoForm));

    InfoForm= UxCreateForm("InfoForm", NO_PARENT);
    UxPutContext(InfoForm, _UxCon);
    UxPutDefaultShell(InfoForm, "topLevelShell");
    InfoFormClose_PBG= UxCreatePushButtonGadget("InfoFormClose_PBG", InfoForm);
    InfoFormDraw= UxCreateDrawingArea("InfoFormDraw", InfoForm);

    _UxToContext(_UxCon);

    Init_InfoForm(InfoForm);

    UxCreateWidget(InfoForm);

    UxPutBottomAttachment(InfoFormClose_PBG, "attach_form");
    UxPutBottomOffset(InfoFormClose_PBG, (int)2);
    UxPutLeftAttachment(InfoFormClose_PBG, "attach_none");
    UxPutLeftOffset(InfoFormClose_PBG, (int)2);
    UxPutRightAttachment(InfoFormClose_PBG, "attach_none");
    UxCreateWidget(InfoFormClose_PBG);
    UxAddCallback(InfoFormClose_PBG, "activateCallback", activateCallback_InfoFormClose_PBG, _UxCon);

    UxPutBottomAttachment(InfoFormDraw, "attach_widget");
    UxPutBottomOffset(InfoFormDraw, (int)2);
    UxPutBottomWidget(InfoFormDraw, "InfoFormClose_PBG");
    UxPutLeftAttachment(InfoFormDraw, "attach_form");
    UxPutLeftOffset(InfoFormDraw, (int)2);
    UxPutRightAttachment(InfoFormDraw, "attach_form");
    UxPutRightOffset(InfoFormDraw, (int)2);
    UxPutTopAttachment(InfoFormDraw, "attach_form");
    UxPutTopOffset(InfoFormDraw, (int)2);
    UxCreateWidget(InfoFormDraw);

    UxRealizeInterface(InfoForm);

    return(InfoForm);
}

static void Init_InfoForm(_Uxsw)
    swidget _Uxsw;
{
    _UxFromContext((_UxCInfoForm *) UxGetContext(_Uxsw));

    UxPutX(InfoForm, (int)700);
    UxPutY(InfoForm, (int)0);
    UxPutWidth(InfoForm, (int)310);
    UxPutHeight(InfoForm, (int)350);
    UxPutDialogTitle(InfoForm, "Block selection Demo -- Help");

    UxPutX(InfoFormClose_PBG, (int)10);
    UxPutY(InfoFormClose_PBG, (int)200);
    UxPutWidth(InfoFormClose_PBG, (int)90);
    UxPutHeight(InfoFormClose_PBG, (int)30);
    UxPutLabelString(InfoFormClose_PBG, "Done");

    UxPutX(InfoFormDraw, (int)10);
    UxPutY(InfoFormDraw, (int)30);
    UxPutWidth(InfoFormDraw, (int)340);
    UxPutHeight(InfoFormDraw, (int)200);
    UxPutBackground(InfoFormDraw, "black");
    UxPutResizePolicy(InfoFormDraw, "resize_none");
}

static void _UxToContext((_UxCInfoForm *) UxGetContext(_Uxsw));
{
    _UxCon= (_UxCInfoForm *) UxGetContext(_UxCon);
    _UxCon->InfoForm       = _UxCon;
    _UxCon->InfoFormClose_PBG= InfoFormClose_PBG;
    _UxCon->InfoFormDraw= InfoFormDraw;
}
```

【図49】

```
#include <stdio.h>
#include <stdlib.h>
#include <X11/Xtrinsic.h>
#include <X11/StringDefs.h>
#include <Xm/Xm.h>
#include <Xm/DrawingA.h>
#include <X11/Xutil.h>
#include <time.h>
#include "block.h"

struct timeval first, second, lapsed;
struct timezone tzp;

/*****************************************************************
 * Start the timing clock
 *****************************************************************/
start_clock ()
{
  gettimeofday (&first, &tzp);
}

/*****************************************************************
 * Stop the timing clock and return the elapsed time in milliseconds.
 *****************************************************************/
double stop_clock ()
{
  double time;

  gettimeofday (&second, &tzp);
  if (first.tv_usec > second.tv_usec) {
      second.tv_usec += 1000000;
      second.tv_usec--;
  }
  lapsed.tv_usec = second.tv_usec - first.tv_usec;
  lapsed.tv_sec  = second.tv_sec - first.tv_sec;
  time = (double)((double)lapsed.tv_usec / (double)1000);
  printf ("**** Elapsed time => %.3f\n msec.", time);
  return time;
}

/*****************************************************************
/* Return the basename of a file without the directory */
/*****************************************************************/
char *extract_base_file_no_dir (name)
  char *name;
{
  char tmp[FILE_LENGTH], *pos;

  strcpy (tmp, name);
  if ((pos = strchr(tmp, '/')) != NULL)
      return pos+1;
  else
      return name;
}

/*****************************************************************
/* Return the basename of a file */
/*****************************************************************/
char *extract_base_file (name)
  char *name;
{
  char tmp[FILE_LENGTH], *pos;

  strcpy (tmp, name);
  if ((pos = strchr(tmp, '.')) != NULL)
      *pos = '\0';
  return tmp;
}

/* ***************************************** */
/* Return -1 if file cannot open */
/* ***************************************** */
int check_file (name)
  char *name;
{
  FILE *fp;

  if ((fp = fopen (name, "r")) == NULL)
      return -1;
  fclose (fp);
  return 1;
}

print_error (mesg)
  char *mesg;
{
  printf ("****Error:  %s\n", mesg);
}

error_create_subproc ()
{
  printf ("****Error:  Cannot create a subprocess!\n");
}

error_subproc_exit_callback ()
{
  printf ("****Error:  Cannot set subprocess exit callback!\n");
}

error_run_subproc ()
{
  printf ("****Error:  Cannot start a subprocess!\n");
}
```

【図50】

```
/*****************************************************************/
/*                                                               */
/*                  File: xs.c                                   */
/*                  Author: Shin-Yuan Wang                       */
/*                  Date : 3-20-92                               */
/*                  Copyright 1992 Canon Information Systems     */
/*                                                               */
/*****************************************************************/
xs_flush(wd)
Widget wd;
{
    /* Flush the queue to X-server */
    XFlush(XtDisplay(wd));
}

/* ---------------------Includes---------------------- */
#include <stdio.h>
#include <Xm/Xm.h>
#include 'color.h'

/* ---------------------Global Var---------------------- */

extern Widget UxTopLevel;
GC drawing_gc;
XGCValues gcv;

void init_graphics(wd)
Widget wd;
{
    drawing_gc = XCreateGC(XtDisplay(wd),
                           DefaultRootWindow(XtDisplay(wd)),
                           0L, (XGCValues *)NULL);
}

int xs_get_color(red, green, blue, wd)
int red, green, blue;
Widget wd;
{
  XColor color;
  Colormap cmap;

  cmap = DefaultColormap(XtDisplay(wd),
                         DefaultScreen(XtDisplay(wd)));

  color.red = red*256;
  color.green = green*256;
  color.blue = blue*256;

  if (XAllocColor(XtDisplay(wd), cmap, &color))
    return (color.pixel);
  printf("Warning: couldn't allocate requested color\n");
  return BlackPixel(XtDisplay(wd), DefaultScreen(XtDisplay(wd)));
}

void xs_set_foreground(color, wd)
int color;
Widget wd;
{
    XSetForeground(XtDisplay(wd), drawing_gc,
                   xs_get_color(color_comp[color][R],
                                color_comp[color][G],
                                color_comp[color][B],
                                wd));
}

void xs_plotxy(wd, x,y)
Widget wd;
int x, y;
{
    XDrawPoint(XtDisplay(wd),XtWindow(wd), drawing_gc, x, y);
}

xs_unplotxy(wd, x, y)
Widget wd;
int x, y;
{

    Arg args[2];
    int background, foreground;

    XtSetArg(args[0],XtNbackground,&background);
    XtSetArg(args[1],XtNforeground,&foreground);
    XtGetValues(wd,args,2);
    XSetForeground(XtDisplay(wd), drawing_gc, background);
    XDrawPoint(XtDisplay(wd), XtWindow(wd), drawing_gc, x, y);
    XSetForeground(XtDisplay(wd), drawing_gc, foreground);
}

xs_clr_win(wd)
Widget wd;
{
  XClearArea(XtDisplay(wd),XtWindow(wd),0,0,0,0,TRUE);
}

xs_draw_box(wd, x,y,width,height)
Widget wd;
int   x, y, width, height;
{
  XDrawRectangle(XtDisplay(wd), XtWindow(wd), drawing_gc, x,y,width,height);
}

xs_string(wd, x,y, str)
Widget wd;
int x, y;
char *str;
{
  XDrawString(XtDisplay(wd), XtWindow(wd), drawing_gc, x, y,str,strlen(str));
}

xs_win_dim(wd, width,height)
Widget wd;
int *width;
int *height;
{
    Arg args[2];
    Dimension win_width,win_height;

    XtSetArg(args[0],XtNwidth,&win_width);
    XtSetArg(args[1],XtNheight,&win_height);
    XtGetValues(wd,args,2);

printf('width = %d length = %d\n', win_width, win_height);
    *width = win_width;
```

【図51】

```
    *height = win_height;
}

xs_draw_line(wd, x1,y1,x2,y2)
Widget wd;
int x1, y1, x2, y2;
{
    XDrawLine(XtDisplay(wd), XtWindow(wd), drawing_gc,  x1, y1, x2, y2);
}

xs_color_point(wd, x, y, color)
Widget wd;
int x, y, color;
{
    Arg args[1];
    int foreground;

    XtSetArg(args[0],XtNforeground,&foreground);
    XtGetValues(wd,args,1);

    XSetForeground(XtDisplay(wd), drawing_gc, color);
    XDrawPoint(XtDisplay(wd), XtWindow(wd), drawing_gc, x, y);
    XSetForeground(XtDisplay(wd), drawing_gc, foreground);
}

xs_color_line(wd, x1, y1, x2, y2, color)
Widget wd;
int x1, y1, x2, y2, color;
{
    Arg args[1];
    int foreground;

    XtSetArg(args[0],XtNforeground,&foreground);
    XtGetValues(wd,args,1);

    XSetForeground(XtDisplay(wd), drawing_gc, color);
    XDrawLine(XtDisplay(wd), XtWindow(wd), drawing_gc, x1, y1, x2, y2);
    XSetForeground(XtDisplay(wd), drawing_gc, foreground);
}

xs_color_box(wd, x1, y1, x2, y2, color)
Widget wd;
int x1, y1, x2, y2, color;
{
    Arg args[1];
    int foreground;

    XtSetArg(args[0],XtNforeground,&foreground);
    XtGetValues(wd,args,1);

    XSetForeground(XtDisplay(wd), drawing_gc, color);
    XDrawRectangle(XtDisplay(wd), XtWindow(wd), drawing_gc, x1, y1, x2, y2);
    XSetForeground(XtDisplay(wd), drawing_gc, foreground);
}

xs_color_str(wd, x, y, str, color)
Widget wd;
int x, y, color;
char *str;
{
    Arg args[1];
    int foreground;

    XtSetArg(args[0],XtNforeground,&foreground);
    XtGetValues(wd,args,1);

    XSetForeground(XtDisplay(wd), drawing_gc, color);
    XDrawString(XtDisplay(wd), XtWindow(wd), drawing_gc, x, y, str, strlen(str));
    XSetForeground(XtDisplay(wd), drawing_gc, foreground);
}

static char **font;
xs_getfont(wd)
Widget wd;
{

    int i, num = 50;

    font = XListFonts(XtDisplay(wd), "*", 50, &num);

    printf("num = %d \n", num);
}

xs_setfont(wd, fid)
Widget wd;
int fid;
{

    XFontStruct *font_info;

    if ((font_info = XLoadQueryFont(XtDisplay(wd), font[fid]))==NULL){
      printf("can not open the font fid = %d \n", fid);
      exit(-1);
    }

    XSetFont(XtDisplay(wd), drawing_gc, font_info->fid);
    printf("fid = %d \n", fid);
}
```

【図52】

```
/*****************************************************************************
 *
 *      Program: cimages.c                              ************
 *      Author:  Shih-Yuan Wang, Chris Sherrick ************
 *      Date:    1-20-92                        ************
 *      Copyright 1992 Canon Information Systems ************
 *
 *****************************************************************************/

#include "blockgeneral.h"
#include "images.h"
#include <calloc.h>

extern unsigned char **matrix();


/*****************************************************************************
 *
 *      Allocate memory for an cimage from memory ************
 *
 *****************************************************************************/

struct images *image_allocate(r, c)
int     r, c;
{

  struct images *page;

  page = (struct images *)malloc(sizeof(struct images));
  if (page == NULL) {
     printf("allocation error: in image_allocate\n");
     exit(1);
  }
  if (r <= 0 || c <= 0)
  {
    printf("illegal cimage size: %d by %d \n", r ,c);
    exit(1);
  }
  page->sizeR = r;                      /* set cimage sizes */
  page->sizeC = c;
  page->pixel = cmatrix(0, page->sizeR-1, 0, page->sizeC-1);
  return page;
}


/*****************************************************************************
 *
 *      Deallocate memory for an cimage from memory ************
 *
 *****************************************************************************/

void image_deallocate(page)
struct images *page;
{
  if ((page->sizeR != 0) && (page->sizeC != 0))
  {
    free_cmatrix(page->pixel, 0, page->sizeR-1, 0, page->sizeC-1);
    page->sizeR = 0;                    /* zero out the sizes */
    page->sizeC = 0;
  }
}


/*****************************************************************************
 *
 *      Read in a byte from a tiff file
 *
 *****************************************************************************/

int getbyte(f, file_position, eoferror)
FILE *f;
long int *file_position;
int  eoferror;
{
  int c1;
  c1 = getc(f);
  if ((c1 == EOF) && (eoferror == 1))
  {
    printf("\n Error: Premature End of File :in getbyte()\n");
    exit(1);
  }
  (*file_position)++;
  return c1;
}

/*****************************************************************************
 *
 *      Read in an integer from a file.  (LSB, MSB)
 *
 *****************************************************************************/
long int getshort(f, fp)
FILE *f;
long int *fp;
{
  int c1, c2;
  long int c3;
  c1 = getbyte(f, fp, 1);
  c2 = getbyte(f, fp, 1);
  c3 = (long int)(c1 + c2*256);
  return c3;
}

/*****************************************************************************
 *
 *      Read in a long integer
 *
 *****************************************************************************/
long int getlong(f, fp)
FILE *f;
long int *fp;
{
  long int c1, c2;
  c1 = getbyte(f, fp);
  c2 = getbyte(f, fp);
  return c1 + c2*65536;
}

/*****************************************************************************
 *
 *      Read in part of a Tiff file             ************
 *
 *****************************************************************************/

struct images *readtiff(filename)
char *filename;
{
  FILE *tagfile;
  long int tagfile_position;    /* Input file                        */
  long int width;               /* The cur file position (in bytes) */
  long int length;              /* Width of picture                 */
  long int black;               /* Length of picture                */
  long int bpa;                 /* Value of black pixel             */
  long int raster_offset;       /* Bytes per strip                  */
  long int numtags, tag, type, len, value;  /* Start of image data  */
  long int i,j;                 /* Tag values                       */
  long int all_width, all_length;
  int v;
  struct images *page;
  ...
  if ((tagfile = fopen(filename, "rb")) == NULL)
```

【図53】

```
    printf("\nError in opening %s \n ",  filename);
    exit(1);
  }
 tagfile_position = 0;

  /* Read in header and make sure it's Intel!  */
  i = get2bytes(tagfile, &tagfile_position);
/* printf("i = %x \n", i);*/
  if (i != 0x4949)
  {
    printf("... Error: TIFF File not in Intel Format\n");
    exit(1);
  }

  /* Read in Magic number                     */
  i = get2bytes(tagfile, &tagfile_position);
/* printf("i = %x \n", i); */
  if (i != 0x002A)
  {
    printf(" BAD Magic Number for TIFF file!!!\n");
    exit(1);
  }

  /* Find First File directory                */
  j = get4bytes(tagfile, &tagfile_position);
 while(tagfile_position < j)
  {
    i = get2bytes(tagfile, &tagfile_position);
  }

  /* Tags in Directory                        */
  numtags = get2bytes(tagfile, &tagfile_position);
/* printf("numtags = %x \n", numtags);   */
  while (--numtags >= 0) {
    tag = get2bytes(tagfile, &tagfile_position);
    type = get2bytes(tagfile, &tagfile_position);
    len = get4bytes(tagfile, &tagfile_position);
    value = get4bytes(tagfile, &tagfile_position);

    if ((len != 1) && ((tag & 32768) == 0))
    {
      printf("Error: Tag Length != 1\n");
      exit(1);
    }
/* printf("tag = %x \n", tag);   */
    switch (tag)
    {
      case 0x00FE:              /* NewSubfileType */
        if (value != 0)
        {
          printf("Can't Deal with Non-Standard NewSubFileType\n");
          exit(1);
        }
        break;
      case 0x00FF:              /* NewSubfileType */
        if (value != 1)
        {
          printf("Subfile Type not full resolution\n");
          exit(1);
        }
        break;
      case 0x0100:              /* Width          */
        width = value;
        break;
      case 0x0101:              /* Length         */
        length = value;
        break;
      case 0x0102:              /* Bits per sample  */
        if (value != 1)
        {
          printf(": Error: Bits/Sample must be 1!");
          exit(1);
        }
        break;
      case 0x0103:              /* Compression      */
        if (value != 1)
        {
          printf("Compression Used! Error!\n");
          exit(1);
        }
        break;
      case 0x0106:              /* Black is         */
        black = value;
        break;
      case 0x0107:              /* Thresholding tag */
        if ((value < 1) || (value > 3))
        {
          printf("Thresholding Tag = Illegal value!\n");
          exit(1);
        }
        break;
      case 0x0111:              /* Offset to Raster Data */
        raster_offset = value;
        break;
      case 0x0117:
        bps = value;
        break;
      case 0x011a:
        break;
      case 0x011b:
        break;
      case 0x0128:
        break;
      default:
        if ((value & 32768) == 0) {
          printf("##### Warning: An unsupported tag appears in the TIF file\n");
        }
        break;
    }
  }

  while(tagfile_position < raster_offset)
  {
    i = get2bytes(tagfile, &tagfile_position);
  }

  /*------------------   READ IN FILE   ------------------------   */
  all_length = (int) ((length+SCALE_RATIO)/SCALE_RATIO)*SCALE_RATIO;
  all_width = (int)((width+SCALE_RATIO)/SCALE_RATIO)*SCALE_RATIO;
/* printf("all_length = %d all_width = %d \n", all_length, all_width); */

  page = image_Allocate((int) all_length, (int) all_width);
/*
printf("page->sizeR = %d page->sizeC = %d \n", page->sizeR, page->sizeC);
printf("black = %d \n", black);
*/
  for (i = 0; i<length; i++)
  {
    for (j = 0; j<width; j++)
    {
      if ((j%8) == 0)
      {
        v = getc(tagfile);
        tagfile_position++;
```

【図54】

```
        if (v == EOF)
        {
          printf("\n Error: Premature End of File found in tiff file\n");
          exit(1);
        }
      }
      if ((v & 128) == black*128)
        page->pixel[i][j] = 0;
      else
        page->pixel[i][j] = 1;
      v = v << 1;
    }
  }

  fclose(tagfile);
  return page;
}
```

【図86】

```
            release_block(tmp1);
            tmp1 = tmp11;
        }
        for (; tmp2 != NULL; ){
            tmp12 = tmp2->next;
            release_block(tmp2);
            tmp2 = tmp12;
        }
        free_rectblock(initial_block);
        return;

}
/*************************************************************************/
/******** Draw text components for display                       ********/
/*************************************************************************/

void block_circumulate(p, width, height, p_ori)
unsigned char **p;
int          width, height;
unsigned char **p_ori;
{

        int i, j;
        /* open file for storing block data */

        original_p = p_ori;
        reduce_factor = SCALE_RATIO;

        for (i = 0; i <= 8; i++)
            for (j = 0; j <= 8; j++)
                direct_statis[i][j] = 0;

        block_file = fopen("block_file.dat", "w");


        text_length = 150.0/SCALE_RATIO;
        total_text = 0;

        /* the root block stands for the whole page image unit */

        root_rectblock = new_block(0,0);
        root_rectblock->width = width;
        root_rectblock->length = height;
        root_rectblock->upper_y = 0;
        root_rectblock->lower_y = height-1;
        root_rectblock->left_x = 0;
        root_rectblock->right_x = width-1;

        /* find blocks inside the page range */
        block_in_range(p, 0, 0,  width-1, height-1, root_rectblock);
printf("get out of block_in_range\n");
        rearrange_rectblock(root_rectblock);
printf("get out of rearrange_rectblock\n");

        text_discriminate(root_rectblock);

printf("get out of text_dis\n");

        firstlevelnontextsearch(p, root_rectblock);

printf("get out of firstlevelnontextsearch\n");
        print_block(1, root_rectblock);


        fclose(block_file);
printf("before get out of block_construct\n");

/*      release_block(root_rectblock);*/
}
```

【図137】

Source Code for

Proportional Spaced Segmentation

【図55】

```
/*****************************************************************************/
/*                                                                           */
/*****************************************************************************/
/*                                                                           */
/*                  File: block_main.c                                       */
/*                  Author: Shin-Yuan Wang                                   */
/*                  Date : 3-30-93                                           */
/*                  Copyright 1993 Canon Information Systems                 */
/*                                                                           */
/*****************************************************************************/
/*                                                                           */
/*****************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <X11/Intrinsic.h>

#include "block_main.h"
#include "blockgeneral.h"
#include "images.h"

extern struct block_rect *get_rectblock();
struct linkblock *linkblock;
struct sectionblock *firstsectionblock;

extern unsigned char **matrix();
extern struct block_image *readtiff();
extern void block_circumscribe();
extern void int_line_block();
extern void int_section_block();
extern struct linkblock *new_linkblock();
extern struct sectionblock *new_sectionblock();
extern void  print_linkblock();
extern void  print_sectionblock();
extern void  free_pair();
extern void  free_rectblock();

int  step = 0;
struct block_image reduce_image;
struct image_pair *p;
float scale_displayform;
unsigned char **p_image;

/* unsigned char **block_matrix; */

/*****************************************************************************/
/*                                                                           */
/*****************************************************************************/
/*                                                                           */
/*                  eliminate the noise                                      */
/*                                                                           */
/*****************************************************************************/

void get_rid_of_pepper(pixel, first_x, first_y, second_x, second_y)
unsigned char **pixel;
int   first_x, first_y, second_x, second_y;
{
    int i, j;

    for (i = first_x; i <= second_x ; i++)
        for (j = first_y ; j < second_y; j++){
            if ((pixel[j][i-1] == 1)&&
                (pixel[j][i+1]==0)&&
                (pixel[j-1][i-1]==0)&&
                (pixel[j-1][i]==0)&&
                (pixel[j-1][i+1]==0)&&
                (pixel[j+1][i-1]== 0)&&
                (pixel[j+1][i]==0)&&
                (pixel[j+1][i+1]==0))
                pixel[j][i] = 0;
            }
}

/*****************************************************************************/
/*                                                                           */
/*****************************************************************************/
/*                                                                           */
/*                  reduce the image                                         */
/*                                                                           */
/*****************************************************************************/

reduce_image(pixel, first_x, first_y, second_x, second_y)
unsigned char **pixel;
int   first_x, first_y, second_x, second_y;
{
    int i, j, k;

    reduce_image.width = page->misc/SCALE_RATIO+2;
    reduce_image.height = page->misc1/SCALE_RATIO+2;

    reduce_image.pixel = cmatrix(0, reduce_image.height-1, 0, reduce_image.width-1);

  /* method 4 */

    for ( i = 0 ; i < reduce_image.height - 2; i++)
        for ( j = 0; j < reduce_image.width - 2 ; j++)
            for ( k = 0; k < TERMINAL; k++)
                if ((page->pixel[y+i*SCALE_RATIO]
                    [x+j*SCALE_RATIO]
                    &(page->pixel[y+yoff[k][0]][x+xoff[k][0]]&&
                    (page->pixel[y+yoff[k][1]][x+xoff[k][1]]&&k>0))
                    {
                        reduce_image.pixel[i+1][j+1] = 1;
                        break;
                    }
}

/*****************************************************************************/
/*                                                                           */
/*****************************************************************************/
/*                                                                           */
/*                  display the specified image matrix into the window       */
/*                                                                           */
/*****************************************************************************/

void display_matrix(p, xmin, ymin, xmax, ymax, width, height, wd, color)
unsigned char **p;
int
xmin, ymin, xmax, ymax, width, height, color;
Widget  wd;

{
    float scale;
    int  x, y, x1, y1;

/*
printf("get into display \n");

printf("width = %d height = %d xmin = %d ymin = %d xmax = %d ymax = %d\n",
        width, height, xmin, ymin, xmax, ymax);
*/
    scale_displayform = (float)(ymax-ymin+1)/(float)(xmax-xmin+1);
    scale = (float)height/(float)width;

/* printf("scale_displayform = %f scale = %f \n", scale_displayform, scale); */
    if (scale_displayform > scale)
        width = (int)((float)height/(float)scale_displayform);
    else
        height = (int)((float)width*scale_displayform);

    scale_displayform = (float)(ymax-ymin+1)/(float)height;
    if (scale_displayform < 1.0){
        scale_displayform  = 1.0;
        width = (xmax-xmin+1);
```

【図56】

```
    height = (ymax-ymin+1);
  }
/*
printf("width = %d height = %d scale_displayform = %f\n",
      width, height, scale_displayform);
*/
  if (color == 0) {
    for ( y = 0; y < height; y++){
      printf("y = %d \n", y);
      for ( x = 0; x < width; x++){
        x1 = (int)((float)x*scale_displayform)+xmin;
        y1 = (int)((float)y*scale_displayform)+ymin;
        if (p[y][x]!=BACK)
          xs_plotxy(wd, x+1, y+1);
      }
    }
    printf("after plot \n");
  } else {
    for ( y = 0; y < height; y++)
      for ( x = 0; x < width; x++){
        x1 = (int)((float)x*scale_displayform)+xmin;
        y1 = (int)((float)y*scale_displayform)+ymin;
        if (p[y][x]!=BACK)
          xs_color_point(wd, x+1, y+1, color);
      }
  }
/*
  xs_flush(wd);
/*   printf("after flush \n"); */
/*
}

/*****************************************************************/
/*                                                               */
/*****************************************************************/

void display_rectblock(firstblock, wd)
struct block_rect *firstblock;
Widget wd;
{

  struct block_rect *tmpb;
  int  x1, y1, x2, y2;
/*
printf("scale_display =%f\n",  scale_displayform);
*/
  if (firstblock->attr&TABLE)
    xs_set_foreground(TEXT_COLOR, wd);
  else
    xs_set_foreground(TEXT_COLOR, wd);
  for (tmpb = firstblock->firstsubblock; tmpb != NULL; tmpb = tmpb->next){
    x1 = (int)((float)tmpb->left_x*scale_displayform);
    y1 = (int)((float)tmpb->upper_y*scale_displayform);
    x2 = (int)((float)tmpb->width*scale_displayform);
    y2 = (int)((float)tmpb->length*scale_displayform);
    if (x1 < 1) x1 = 1;
    if (y1 < 1) y1 = 1;
    xs_draw_box(wd, x1, y1, x1+x2-1, y1+y2-1);
  }

  for (tmpb = firstblock->firstmemberblock; tmpb != NULL; tmpb = tmpb->next){
    x1 = (int)((float)tmpb->left_x*scale_displayform);
    y1 = (int)((float)tmpb->upper_y*scale_displayform);
    x2 = (int)((float)tmpb->width*scale_displayform);
    y2 = (int)((float)tmpb->length*scale_displayform);
    if (x1 < 1) x1 = 1;
    if (y1 < 1) y1 = 1;
    xs_draw_box(wd, x1, y1, x1+x2-1, y1+y2-1);
  }

  if (x2 < 1) x2 = 1;
  if (y2 < 1) y2 = 1;

  if (tmpb->attr&TABLE)
    xs_set_foreground(TABLE_COLOR, wd);
  else if (tmpb->attr&VALID)
    xs_set_foreground(LINE_COLOR, wd);
  else if (tmpb->attr&INVALID)
    xs_set_foreground(INVALID_COLOR, wd);
  else if (tmpb->attr&PICTURE)
    xs_set_foreground(PICTURE_COLOR, wd);
  else
    xs_set_foreground(UNKNOWN_COLOR, wd);
  xs_draw_box(wd, x1, y1, x1+x2-1, y1+y2-1);
  if (tmpb->attr&PICTURE)
    xs_set_foreground(FRAME_COLOR, wd);
  else if (tmpb->attr&PICTURE)
    xs_set_foreground(PICTURE_COLOR, wd);
  else if (tmpb->attr&UNKNOWN)
    xs_set_foreground(UNKNOWN_COLOR, wd);
  xs_draw_box(wd, x1, y1, x1+x2-1, y1+y2-1);
  if ((x1 >= 3)&&( y1 >= 3))
    xs_draw_box(wd, x1+1, y1+1, x1+x2-2, y1+y2-2);
  if (tmpb->attr&(PICTURE|VSOLID|HSOLID))
    continue;
  else
    display_rectblock(tmpb, wd);
 }
}

/*****************************************************************/
/*                                                               */
/*****************************************************************/

void display_splines(wd)
Widget wd;
{
  int x1, y1, x, y;

  xs_set_foreground(LINEDATA_COLOR, wd);

  for (x = 0; x < reduce_image.width; x++)
    for (y = 0; y < reduce_image.height; y++)
      if (reduce_image.pixel[y][x]==MARK) {
        x1 = (int)((float)x/scale_displayform);
        y1 = (int)((float)y/scale_displayform);
        xs_plotxy(wd, x1+1, y1+1);
      }
}

/*****************************************************************/
/*                                                               */
/*****************************************************************/

void display_lineblock(firstline, wd)
struct lineblock *firstline;
Widget wd;
{

  struct lineblock *tmpl;
  int  x1, y1, x2, y2, l1;
/*
printf("get into linedis \n");
*/
  if (firstline->attr&TABLE)
    xs_set_foreground(TABLETEXT_COLOR, wd);
  else
    xs_set_foreground(TEXT_COLOR, wd);
```

【図57】

```
    for (tmpl = firstline->firstsubline; tmpl != NULL; tmpl = tmpl->next){
        x1 = (int)((float)tmpl->left_x/scale_displayform);
        y1 = (int)((float)tmpl->upper_y/scale_displayform);
        w1 = (int)((float)tmpl->width/scale_displayform);
        l1 = (int)((float)tmpl->length/scale_displayform);
        if (w1 < 1) w1 = 1;
        if (l1 < 1) l1 = 1;
        xs_draw_box(wd, x1, y1, w1-1, l1-1);
        if (tmpl->att&CAPTION)
          xs_draw_line(wd, x1, y1, x1+5, y1+5);
    }

    for (tmpl = firstline->firstnontextline; tmpl != NULL; tmpl = tmpl->next){
        x1 = (int)((float)tmpl->left_x/scale_displayform);
        y1 = (int)((float)tmpl->upper_y/scale_displayform);
        w1 = (int)((float)tmpl->width/scale_displayform);
        l1 = (int)((float)tmpl->length/scale_displayform);
        if (w1 < 1) w1 = 1;
        if (l1 < 1) l1 = 1;

        if (tmpl->att&TABLE)
            xs_set_foreground(TABLE_COLOR, wd);
        else if (tmpl->att&(VSOLID|HSOLID))
                xs_set_foreground(LINE_COLOR, wd);
        else if (tmpl->att&HPICTURE)
                xs_set_foreground(HPICTURE_COLOR, wd);
             else if (tmpl->att&FRAME)
                     xs_set_foreground(FRAME_COLOR, wd);
                  else if (tmpl->att&PICTURE)
                          xs_set_foreground(PICTURE_COLOR, wd);
                       else if (tmpl->att&UNKNOWN)
                               xs_set_foreground(UNKNOWN_COLOR, wd);
         xs_draw_box(wd, x1, y1, w1-1, l1-1);
         if ((w1 >= 3)&&( l1 >= 3))
           xs_draw_box(wd, x1+1, y1+1, w1-3, l1-3);
         if ((tmpl->att&(HPICTURE|VSOLID|HSOLID|PICTURE))||
                                          (firstline->att&TABLE))
            continue;
         else
           display_lineblock(tmpl, wd);
    }
}

/*****************************************************************/
/********* draw one side of block                        *********/
/*****************************************************************/

void draw_side(section, wd, color, side)
struct sectionblock *section;
Widget wd;
int    color, side;
{
       int x1, x2, y1, y2;

       x1 = (int)((float)section->left_x/scale_displayform);
       x2 = (int)((float)section->right_x/scale_displayform);
       y1 = (int)((float)section->upper_y/scale_displayform);
       y2 = (int)((float)section->lower_y/scale_displayform);

       xs_set_foreground(color, wd);
       switch(side){
           case 1:  xs_draw_line(wd, x1, y1-2, x2, y1-2);
                    xs_draw_line(wd, x1, y1-1, x2, y1-1);
                    break;
           case 2:  xs_draw_line(wd, x1-2, y1, x1-2, y2);
                    xs_draw_line(wd, x1-1, y1, x1-1, y2);
                    break;
           case 3:  xs_draw_line(wd, x1, y2+2, x2, y2+2);
                    xs_draw_line(wd, x1, y2+1, x2, y2+1);
                    break;
           case 4:  xs_draw_line(wd, x1+2, y1, x1+2, y2);
                    xs_draw_line(wd, x1+1, y1, x1+1, y2);
                    break;
       }
}

/*****************************************************************/
/********* Display the lineblock                                 */
/*****************************************************************/


void display_sectionblock(firstsection, wd)
struct sectionblock *firstsection;
Widget wd;
{

    struct sectionblock *tmps;
    int    x1, y1, w1, l1, x2, y2;

    if (firstsection->att&TABLE)
       xs_set_foreground(TABLETEXT_COLOR, wd);
    else
       xs_set_foreground(TEXT_COLOR, wd);
    for (tmps = firstsection->firstsubsection; tmps != NULL; tmps = tmps->next){
        x1 = (int)((float)tmps->left_x/scale_displayform);
        y1 = (int)((float)tmps->upper_y/scale_displayform);
        w1 = (int)((float)tmps->width/scale_displayform);
        l1 = (int)((float)tmps->length/scale_displayform);
        if (w1 < 1) w1 = 1;
        if (l1 < 1) l1 = 1;
        xs_draw_box(wd, x1, y1, w1-1, l1-1);
        if (tmps->att&TITLE){
           if ((w1 >= 3)&&( l1 >= 3))
              xs_draw_box(wd, x1+1, y1+1, w1-3, l1-3);
        }
/*
        if (tmps->att&CAPTION)
          xs_draw_line(wd, x1, y1, x1+5, y1+5);
*/
    }
    for (tmps=firstsection->firstnontextsection; tmps != NULL; tmps = tmps->next){
        x1 = (int)((float)tmps->left_x/scale_displayform);
        y1 = (int)((float)tmps->upper_y/scale_displayform);
        w1 = (int)((float)tmps->width/scale_displayform);
        l1 = (int)((float)tmps->length/scale_displayform);
        if (w1 < 1) w1 = 1;
        if (l1 < 1) l1 = 1;

        if (tmps->att&TEXT)
           xs_set_foreground(TEXT_COLOR, wd);
        else if (tmps->att&TABLE)
                xs_set_foreground(TABLE_COLOR, wd);
        else if (tmps->att&(VSOLID|HSOLID))
                xs_set_foreground(LINE_COLOR, wd);
        else if (tmps->att&HPICTURE)
                xs_set_foreground(HPICTURE_COLOR, wd);
             else if (tmps->att&FRAME)
                     xs_set_foreground(FRAME_COLOR, wd);
                  else if (tmps->att&PICTURE)
                          xs_set_foreground(PICTURE_COLOR, wd);
                       else if (tmps->att&UNKNOWN)
                               xs_set_foreground(UNKNOWN_COLOR, wd);
```

```
        xs_draw_box(wd, x1, y1, w1-1, l1-1);
        if ((tmps->att&TEXT2)==TEXT){
           x2 = (int)((float)tmps->right_x/scale_displayform);
           y2 = (int)((float)tmps->lower_y/scale_displayform);
           xs_draw_line(wd, x1, y1, x1, y2);
/*           xs_string(wd, x1, y1, "2"); */
        }
        else if ((tmps->att&(TEXT|RECTEST))==(TEXT|RECTEST)){
              x2 = (int)((float)tmps->right_x/scale_displayform);
              y2 = (int)((float)tmps->lower_y/scale_displayform);
              xs_draw_line(wd, x1, y1, x2, y2);
/*              xs_string(wd, x1, y1, "p"); */
           }
           else if ((tmps->att&TITLE)&&(w1 >= 3)&&( l1 >= 3))
                   xs_draw_box(wd, x1+1, y1+1, w1-3, l1-3);
        if ((tmps->att&(HSOLID|TEXT))==(HSOLID|TEXT)){
           if (tmps->att&UP)
              draw_side(tmps, wd, LINE_COLOR, 1);
           else if (tmps->att&DOWN)
              draw_side(tmps, wd, LINE_COLOR, 3);
         } else if ((tmps->att&(VSOLID|TEXT))==(VSOLID|TEXT)){
                   if (tmps->att&UP)
                      draw_side(tmps, wd, LINE_COLOR, 2);
                   else if (tmps->att&DOWN)
                      draw_side(tmps, wd, LINE_COLOR, 4);
                   }


        if (tmps->att&(HPICTURE|VSOLID|HSOLID|PICTURE))
           continue;
        else
           display_sectionblock(tmps, wd);
     }
}

/*********************************************************************/
/********  free rectblock                                  **********/
/*********************************************************************/

void free_block(firstblock)
struct block_rect *firstblock;
{
    struct block_rect *tmp1, *tmp2;
    struct whiteblock *tmpw1, *tmpw2;
    int    i;

/*
printf("get int free rect = %d x = %d y = %d  \n", firstblock->index,
                        firstblock->left_x, firstblock->upper_y);
*/

    for (tmp1 = firstblock->firstsubblock; tmp1 != NULL; ){
/* printf("tmp 1 = %d \n", tmp1); */
       tmp2 = tmp1->next;
       free_rectblock(tmp1);
/* printf("next tmp2 = %d\n" , tmp2); */
       tmp1 = tmp2;
    }
    for ( tmp1 = firstblock->firstcontextblock; tmp1 != NULL; ){
       tmp2 = tmp1->next;
       free_block(tmp1);
/* printf("free next = %d \n", tmp2); */
       tmp1 = tmp2;
    }

    if (firstblock->firstwhite != NULL){
/* printf("firstwhite = %d \n", firstblock->firstwhite); */
          for (tmpw1 = firstblock->firstwhite; tmpw1 != NULL; ){
/* printf("subwhite = %d \n", tmpw1->numsubwhite); */
             tmpw2 = tmpw1->next;
             if (tmpw1->numsubwhite != 0){
                for ( i = 0; i < tmpw1->numsubwhite; i++){
                    free_pair(tmpw1->subwhite[i].pairright,
                              tmpw1->subwhite[i].pairleft,
                              tmpw1->subwhite[i].upper_y,
                              tmpw1->subwhite[i].lower_y);
                }
                free((struct whiteblock *)tmpw1->subwhite);
             }
             free_white(tmpw1);
/* printf("after free white, next = %d \n", tmpw2);  */
             tmpw1 = tmpw2;
          }
/* printf("after release white \n"); */
       }

    free_rectblock(firstblock);
}


/*********************************************************************/
/********  free_section                                    **********/
/*********************************************************************/

void free_section(firstsection)
struct sectionblock *firstsection;
{
    struct sectionblock *tmps1, *tmps2;
    struct lineblock    *tmpl1, *tmpl2;
    struct block_rect   *tmpr1, *tmpr2;
/*
 printf("get into free section \n");
*/
    for (tmps1 = firstsection->firstsubsection; tmps1 != NULL; ){
        tmps2 = tmps1->next;
/*
 printf("section = %d section = %d \n", tmps1, tmps1->index);
*/
        for (tmpl1 = tmps1->firstlineblock; tmpl1 != NULL; ){
            tmpl2 = tmpl1->next;
/*
printf("tmpl1 = %d line = %d first = %d current %d \n", tmpl1,
                        tmpl1->index, tmpl1->first_block->index,
                    tmpl1->current_block->index);
*/

            for (tmpr1 = tmpl1->first_block; tmpr1 != NULL; ){
               tmpr2 = tmpr1->next;
               free_block(tmpr1);
/*
printf("after free rect, tmpr2 = %d tmpl1 = %d \n", tmpr2, tmpl1);
*/
               tmpr1 = NULL;
               tmpr1 = tmpr2;
            }
            tmpl1->first_block = NULL;
/*
printf("free line tmpl1 = %d tmpl1->index = %d under-id r_cap=%d\n",
        tmpl1, tmpl1->index, tmpl1->under_caption, tmpl1->r_caption);
*/
            free((struct lineblock *)tmpl1);
/*
printf("after free line tmpl2 = %d \n", tmpl2);
*/
```

【図59】

```
        tmpl = NULL;
        tmp2 = tmp2;
    }
/*
printf("free section tmp1 = %d \n", tmp1);
*/
    free((struct sectionblock *)tmp1);
    tmp1 = NULL;
    tmp1 = tmp2;
}
/*
printf("free section \n");
*/
    for (tmp1 = firstsection->firstnontextsection; tmp1 != NULL; ){
/*
printf("nontext section = %d \n", tmp1->index);
*/
        tmp2 = tmp1->next;
        for (tmp3 = tmp1->firstlineblock; tmp3 != NULL; ){
            tmp4 = tmp3->next;
/*
printf("line = %d first = %d current = %d \n", tmp3->index, tmp3->first_block->signdata[0]->index,
x;
                tmp3->current_block->index);
*/
            for (tmp2 = tmp3->first_block; tmp2 != NULL; ){
                tmp2 = tmp2->next;
                free_block(tmp2);
                tmp2 = NULL;
                tmp3 = tmp4;
            }
            tmp3->first_block = NULL;
/*
printf("nontext free line tmp3 = %d tmp3->index = %d under=%d x_cap=%d\n",
    tmp3, tmp3->index, tmp3->under_caption, tmp3->x_caption);
*/
            free((struct lineblock *)tmp3);
            tmp3 = NULL;
/*
printf("free line \n");
*/
        tmp1 = tmp2;
        }
        tmp1->firstlineblock = NULL;
        free_section(tmp1);
        tmp1 = NULL;
/*
printf("free section \n");
*/
    tmp1 = tmp2;
    }
    free((struct sectionblock *)firstsection);
}

/*.....*/
/*..... Open diff file and reduce the size by 3 .....*/
/*.....*/

void open_diff_file (filename)
char filename[];
{

printf("step = %d \n", step);
    if (step != 0){
                free_matrix(reduce_image.pixel, 0, reduce_image.height-1,
                                                0, reduce_image.width-1);

printf("before  free section \n");
                if (step>SP_COND){
                    free_matrix(D_image, 0, reduce_image.height-1,
                                                0, reduce_image.width-1);

                    free_section(firstsectionblock);
printf("after free section \n");
                }

                image_deallocate(page);
                step = 0;
            }
printf("unloading ...");
page = readdiff(filename);
printf("Done. \n");
get_pid_of_paper(page->pixel, 1.1, page->size0-2, page->size1-2);

printf("size = %d size0 = %d size1 = %d \n", page->size, page->size0,
                create_reduce_image();
                step = SP_LEN3|SP_REDUCE;
            }
        }

/*.....*/
/*..... get text block into text display .....*/
/*.....*/

void get_text(tmpsec1, p, wd, color)
struct sectionblock *tmpsec1;
unsigned char *p;
Widget wd;
int   color;
{
    struct lineblock *tmp1;
    struct block_rect *tmpr;
    int x, y, x1, y1;

    xs_set_foreground(color, wd);
    for (tmp1 = tmpsec1->firstlineblock; tmp1 != NULL; tmp1 = tmp1->next){
        for (tmpr = tmp1->first_block; tmpr != NULL; tmpr = tmpr->next){
            for (y = tmpr->upper_y; y <= tmpr->lower_y; y++){
                for (x = tmpr->left_x; x <= tmpr->right_x; x++){
                    if (reduce_image.pixel[y][x]!=0){
                        p[y][x] = TEXT_BLOCK;
                        y1 = (int)((float)y/scale_displayform);
                        x1 = (int)((float)x/scale_displayform);
                        xs_plot_y(wd, x1, y1);
                    }
                }
            }
        }
    }
}

/*.....*/
/*..... get text into table text display .....*/
/*.....*/

void get_tabletext(tmpsec2, p, wd)
struct sectionblock *tmpsec2;
unsigned char *p;
Widget wd;
{
    int   x, y, x1, y1;
```

【図60】

```
        xs_set_foreground(TABLETEXT_COLOR, wd);
        for (y = tmpsec2->upper_y; y <= tmpsec2->lower_y; y++)
            for (x = tmpsec2->left_x; x <= tmpsec2->right_x; x++){
                if ((reduce_image.pixel[y][x]&0x01)&&(p[y][x]==0)){
                    p[y][x] = TEXT_BLOCK;
                    y1 = (int)((float)y/scale_displayform);
                    x1 = (int)((float)x/scale_displayform);
                    xs_ploxy(wd, x1, y1);
                }
            }
    }

/*************************************************************************/
/*********  get picture into picture display                     *********/
/*************************************************************************/

void get_picture(tmpsec2, p, atr, color, wd)
struct sectionblock *tmpsec2;
unsigned char **p;
int atr, color;
Widget wd;
{
    int x, y, x1, y1;

    xs_set_foreground(color, wd);
    for (y = tmpsec2->upper_y; y <= tmpsec2->lower_y; y++)
        for (x = tmpsec2->left_x; x <= tmpsec2->right_x; x++){
            if ((reduce_image.pixel[y][x]&0x01)&&(p[y][x]==0)){
                p[y][x] = atr;
                y1 = (int)((float)y/scale_displayform);
                x1 = (int)((float)x/scale_displayform);
                xs_ploxy(wd, x1, y1);
            }
        }
}

/*************************************************************************/
/*********  get picture into picture display                     *********/
/*************************************************************************/

void get_linepicture(tmpsec2, p, wd)
struct sectionblock *tmpsec2;
unsigned char **p;
Widget wd;
{
    struct block_rect *tmpr;
    int   x, y, x1, y1, pair;
/* printf("get into line picture\n"); */

    xs_set_foreground(PICTURE_COLOR, wd);

/*
printf("sec - %d x=%d y - %d \n", tmpsec2->index, tmpsec2->upper_y);
printf("line index - %d x=%d y - %d \n", tmpsec2->firstlineblock->index, tmpsec2->fi
rstlineblock->left_x,
tmpsec2->firstlineblock->upper_y);
*/
    tmpr = tmpsec2->firstlineblock->first_block;
/* printf("sec index - %d x=%d y - %d \n", tmpr->index,  tmpr->left_x, tmpr->upper_y
);
*/
    for (y = tmpr->upper_y; y <= tmpr->lower_y; y++)
        for (pair = 1; pair <= tmpr->pairright[y][0]; pair++)
            for (x = tmpr->pairleft[y][pair]; x <= tmpr->pairright[y][pair]; x++)
                if (reduce_image.pixel[y][x]&0x01){
                    p[y][x] = TEXT_BLOCK;
                    y1 = (int)((float)y/scale_displayform);
                    x1 = (int)((float)x/scale_displayform);
                    xs_ploxy(wd, x1, y1);
                }
/* printf("get out line picture\n"); */
}

/*************************************************************************/
/*********  get picture into picture display                     *********/
/*************************************************************************/

void get_line(tmpsec2, p, atr, color, wd)
struct sectionblock *tmpsec2;
unsigned char **p;
int atr, color;
Widget wd;
{
    int x, y, x1, y1;

    xs_set_foreground(color, wd);
    for (y = tmpsec2->upper_y; y <= tmpsec2->lower_y; y++)
        for (x = tmpsec2->left_x; x <= tmpsec2->right_x; x++){
            if ((reduce_image.pixel[y][x]&0x01)){
                p[y][x] = atr;
                y1 = (int)((float)y/scale_displayform);
                x1 = (int)((float)x/scale_displayform);
                xs_ploxy(wd, x1, y1);
            }
        }
}

/*************************************************************************/
/*********  get picture into picture display                     *********/
/*************************************************************************/

void get_picture(tmpsec2, p, atr, color, wd)
struct sectionblock *tmpsec2;
unsigned char **p;
int atr, color;
Widget wd;
{
    int x, y, x1, y1;

    xs_set_foreground(color, wd);
    for (y = tmpsec2->upper_y; y <= tmpsec2->lower_y; y++)
        for (x = tmpsec2->left_x; x <= tmpsec2->right_x; x++){
            if ((reduce_image.pixel[y][x]&0x01)){
                p[y][x] = atr;
                y1 = (int)((float)y/scale_displayform);
                x1 = (int)((float)x/scale_displayform);
                xs_ploxy(wd, x1, y1);
            }
        }
}

/*************************************************************************/
/*********  separate image                                       *********/
/*************************************************************************/

void separate_image(firstsection, wd)
struct sectionblock *firstsection;
Widget wd;
{
    struct sectionblock *tmpsec2, *tmpsec3, *tmpsec4;
    int   x, y;

    for (tmpsec4 = firstsection->firstsubsection; tmpsec4 != NULL;
                                                  tmpsec4 = tmpsec4->next)
        get_text(tmpsec4, h_image, wd, TEXT_COLOR);

    for (tmpsec4 = firstsection->firstcontextsection; tmpsec4 != NULL;
                                                  tmpsec4 = tmpsec4->next){
        if (tmpsec4->att&TEXT){
            get_text(tmpsec4, h_image, wd, TEXT_COLOR);
            if ((tmpsec4->att&HEADID)){
                if (tmpsec4->att&UP)
                    draw_side(tmpsec4, wd, LINE_COLOR, 1);
                else if (tmpsec4->att&DOWN)
                    draw_side(tmpsec4, wd, LINE_COLOR, 3);
            } else if (tmpsec4->att&VOLID){
                if (tmpsec4->att&UP)
                    draw_side(tmpsec4, wd, LINE_COLOR, 2);
                else if (tmpsec4->att&DOWN)
                    draw_side(tmpsec4, wd, LINE_COLOR, 4);
```

【図61】

```
    }
  }
  else if (tmpsecl->attr&(VSOKUIDINSOUID)
      get_line(tmpsecl, b_image, LINE_BLOCK, LINE_COLOR, wd);
      else if (tmpsecl->attr&HPICTURE)
          pm_picture(tmpsecl, b_image,
                     HPICTURE_BLOCK, HPICTURE_COLOR, wd);
      else if (tmpsecl->attr&TABLE){
          for (tmpsec2 = tmpsecl->firstsubsection; tmpsec2 != NULL;
                                  tmpsec2 = tmpsec2->next)
              get_tabletext(tmpsec2, b_image, wd);
          tmpsecl = tmpsecl->firstsubsection;
          separate_image(tmpsecl, wd);
          tmpsecl->firstsubsection = NULL;
          tmpsecl->firstsubsection = tmpsec2;
          get_picture(tmpsecl, b_image, TABLE_BLOCK,
                                  TABLE_COLOR, wd);
      } else if (tmpsecl->attr&FRAME){
          separate_image(tmpsecl, wd);
          get_picture(tmpsecl, b_image, FRAME_BLOCK,
                                  FRAME_COLOR, wd);
      }
      else if (tmpsecl->attr&PICTURE){
          get_linkpicture(tmpsecl, b_image, wd);
          for (tmpsec2 = tmpsecl->firstsubsection;
                  tmpsec2 != NULL; tmpsec2 = tmpsec2->next)
              get_tabletext(tmpsec2, b_image, wd, TEXT_COLOR);
      } else if (tmpsecl->attr&UNKNOWN)
          get_picture(tmpsecl, b_image,
                      UNKNOWN_BLOCK, UNKNOWN_COLOR, wd);
  }
}

/*****************************************************************/
/********** block processing                          ************/
/*****************************************************************/

void block_process(wd, width, height)
Widget wd;
int    width, height;
{
    int yes;

/*
printf("get into process \n");
*/
    init_graphics(wd);
    xs_clr_win(wd);

    display_matrix(reduce_image.pixel, 0, 0, reduce_image.width-1,
                   reduce_image.height-1, width-2, height-2, wd, 0);

    if (!(stepSF_CONN)){
        for (yes = 0; yes == 0; ) {
            printf("go for connected block generation? (1: yes, 0: no) ");
            fscanf(stdin, "%d", &yes);
        }

    printf("step 1 .....\n");
    block_circumscribe(reduce_image.pixel, reduce_image.width,
                       reduce_image.height, page->pixel);
    xs_clr_win(wd);
    display_rectblock(root_rectblock, wd);
    xs_flush(wd);

    for (yes = 0 ; yes == 0; ) {
        printf("go for line block generation? (1: yes, 0: no) ");
        fscanf(stdin, "%d", &yes);
    }

    printf("step 2 .....\n");
    firstlineblock = new_lineblock(0,0);

    ini_line_block(firstlineblock, root_rectblock, reduce_image.width,
                   page->black, reduce_image.pixel);

    print_lineblock(firstlineblock);

    xs_clr_win(wd);
    display_lineblock(firstlineblock, wd);

    display_gapline(wd);

    xs_flush(wd);

    for (yes = 0 ; yes == 0; ) {
        printf("go for section block generation? (1: yes, 0: no) ");
        fscanf(stdin, "%d", &yes);
    }

    printf("step 3 .....\n");
    firstsectionblock = new_sectionblock(0,0);
    ini_section_block(firstsectionblock,firstlineblock, reduce_image.pixel,
                      reduce_image.width);

    print_sectionblock(firstsectionblock);

    step |= SP_CONN;
  }
}

    xs_clr_win(wd);
    display_sectionblock(firstsectionblock, wd);
    xs_flush(wd);

    for (yes = 0 ; yes == 0; ) {
        printf("go for color image generation? (1: yes, 0: no) ");
        fscanf(stdin, "%d", &yes);
    }

    xs_clr_win(wd);
    b_image = cmatrix(0, reduce_image.height-1, 0, reduce_image.width-1);

    separate_image(firstsectionblock, wd);
    xs_flush(wd);
}

/*****************************************************************/
/********** color display                             ************/
/*****************************************************************/

void color_reference(wd, width, height)
Widget wd;
```

【図６３】

```
/*****************************************************************************/
/*                                                                           */
/*  File:  nrmalloc.c                                                        */
/*  Author:  Shin-Yuan Wang                                                  */
/*  Date:  10-2-91                                                           */
/*  Copyright 1992 Canon Information Systems                                 */
/*                                                                           */
/*****************************************************************************/
#include <malloc.h>
#include <stdio.h>

void nrerror(error_text)
char error_text[];
{
    void exit();

    fprintf(stderr, "%s\n", error_text);
    exit(1);
}

/* Allocate a float vector with range [nl..nh] */
float *vector(nl, nh)
int nl, nh;
{
    float *v;

    v = (float *) malloc((unsigned)(nh-nl+1)*sizeof(float));
    if (!v) nrerror("allocation failure in vector()");
    return v-nl;
}

/* Allocate a unsigned char with range [nl..nh] */
unsigned char *cvector(nl, nh)
int nl, nh;
{
    unsigned char *v;

    v = (unsigned char *) malloc((unsigned)(nh-nl+1)*sizeof(unsigned char));
    if (!v) nrerror("allocation failure in cvector()");
    return v-nl;
}

/* Allocate a int vector with range [nl..nh] */
int *ivector(nl, nh)
int nl, nh;
{
    int *v;
    int i;

    v = (int *) malloc((unsigned)(nh-nl+1)*sizeof(int));
    if (!v) nrerror("allocation failure in ivector()");
    v -= nl;
    for (i = nl; i <=nh; i++)
        v[i] = 0;
    return v;
}

/* Allocate a double vector with range [nl..nh] */
double *dvector(nl, nh)
int nl, nh;
{
    double *v;

    v = (double *) malloc((unsigned)(nh-nl+1)*sizeof(double));
    if (!v) nrerror("allocation failure in dvector()");
    return v-nl;
}

/* Allocate a character matrix with range [nrl..nrh][ncl..nch] */
unsigned char **cmatrix(nrl, nrh, ncl, nch)
int nrl, nrh, ncl, nch;
{
    int i, j;
    unsigned char **m;

    m=(unsigned char **)malloc((unsigned)(nrh-nrl+1)*sizeof(unsigned char *));
    if (!m) nrerror("allocation failure 1 in cmatrix()");
        m -= nrl;

    for (i = nrl; i <= nrh; i++) {
    m[i]=(unsigned char *) malloc((unsigned)(nch-ncl+1)*sizeof(unsigned char));
    if (!m[i]) nrerror("allocation failure 2 in cmatrix()");
    m[i] -= ncl;
        for (j = ncl; j <= nch; j++)
            m[i][j]= 0;
    }

    return m;
}

/* Allocate a float matrix with range [nrl..nrh][ncl..nch] */
float **matrix(nrl, nrh, ncl, nch)
int nrl, nrh, ncl, nch;
{
    int i;
    float **m;

    m = (float **) malloc((unsigned)(nrh-nrl+1)*sizeof(float *));
    if (!m) nrerror("allocation failure 1 in matrix()");
    m -= nrl;

    for (i = nrl; i <= nrh; i++) {
        m[i] = (float *) malloc((unsigned)(nch-ncl+1)*sizeof(float));
        if (!m[i]) nrerror("allocation failure 2 in matrix()");
        m[i] -= ncl;
    }
    return m;
}

/* Allocate an int matrix with range [nrl..nrh][ncl..nch] */
int **imatrix(nrl, nrh, ncl, nch)
int nrl, nrh, ncl, nch;
{
    int i;
    int **m;

    m = (int **) malloc((unsigned)(nrh-nrl+1)*sizeof(int *));
    if (!m) nrerror("allocation failure 1 in matrix()");
    m -= nrl;

    for (i = nrl; i <= nrh; i++) {
        m[i] = (int *) malloc((unsigned)(nch-ncl+1)*sizeof(int));
        if (!m[i]) nrerror("allocation failure 2 in matrix()");
        m[i] -= ncl;
    }
    return m;
```

【図64】

```
}

/* Allocate a double matrix with range [nrl..nrh][ncl..nch]. */
double **dmatrix(nrl, nrh, ncl, nch)
int nrl, nrh, ncl, nch;
{
        int i;
        double **m;

        m = (double **) malloc((unsigned)(nrh-nrl+1)*sizeof(double *));
        if (!m) nrerror("allocation failure 1 in matrix()");
        m -= nrl;

        for (i = nrl; i <= nrh; i++){
          m[i] = (double *) malloc((unsigned)(nch-ncl+1)*sizeof(double));
          if (!m[i]) nrerror("allocation failure 2 in matrix()");
          m[i] -= ncl;
        }
        return m;
}

/* deallocate a float vector */
void free_vector(v, nl, nh)
float *v;
int nl, nh;
{
        free((float*)(v+nl));
}

/* deallocate an int vector */
void free_ivector(v, nl, nh)
int *v, nl, nh;
{
        free((int*)(v+nl));
}

/* deallocate a double vector */
void free_dvector(v, nl, nh)
double *v;
int nl, nh;
{
        free((double*)(v+nl));
}

/* deallocate a float matrix */
void free_matrix(m, nrl, nrh, ncl, nch)
float **m;
int nrl, nrh, ncl, nch;
{
        int i;

        for (i = nrh; i >= nrl; i--) free((float*) (m[i]+ncl));
        free((float*) (m+nrl));
}

/* deallocate an int matrix */
void free_imatrix(m, nrl, nrh, ncl, nch)
int **m;
int nrl, nrh, ncl, nch;
{
        int i;

        for (i = nrh; i >= nrl; i--) free((int*) (m[i]+ncl));
        free((int*) (m+nrl));
}

/* deallocate an char matrix */
void free_cmatrix(m, nrl, nrh, ncl, nch)
unsigned char **m;
int nrl, nrh, ncl, nch;
{
        int i;

        for (i = nrh; i >= nrl; i--) free((char*) (m[i]+ncl));
        free((char*) (m+nrl));
}

/* deallocate a double matrix */
void free_dmatrix(m, nrl, nrh, ncl, nch)
double **m;
int nrl, nrh, ncl, nch;
{
        int i;

        for (i = nrh; i >= nrl; i--) free((double*) (m[i]+ncl));
        free((double*) (m+nrl));
}
```

【図65】

```
/*****************************************************************/
/*                                                               */
/*      File: block_construct.c                                  */
/*      Author: Shin-Yuan Wang                                   */
/*      Date : 1-15-92                                           */
/*                                                               */
/*****************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
#include "blockgeneral.h"
#include "blockrect.h"
#include "read.h"

extern    void xs_draw_box();
void      exit();
struct    block_rect *root_rectblock;
int       total_text;
float     text_length;

FILE      *fopen(), *fp, *block_file;
char      *locatext[4] = {"skip", "left", "right", "ignore"};

int       direct_stats[9][9];
int       remove_factor;
unsigned char **original_b;

/*****************************************************************/
/*                                                               */
/*      Maximum of two integer                                   */
/*                                                               */
/*****************************************************************/
int max(a, b)
int a, b;
{
    if (a >= b) return a;
    else return b;
}

/*****************************************************************/
/*                                                               */
/*      avg and std of an integer array                          */
/*                                                               */
/*****************************************************************/
void avg_std(a, no, avg, std)
int *a, no;
float *avg, *std;
{
    int i, total;
    long total_square;

    total = 0; total_square = 0;
    *avg = 0.0; *std = 0.0;
    for ( i = 0; i < no; i++){
        total += a[i];
        total_square += (a[i]*a[i]);
    }
    *avg = (float)(total)/(float)no;
    *std = (float)((double)total_square/(double)no)-(*avg)*(*avg);
}

/*****************************************************************/
/*                                                               */
/*      Allocate memory for block_rect                           */
/*                                                               */
/*****************************************************************/

struct block_rect *new_block(nl, nh)
int nl, nh;
{
    int i;
    struct block_rect *v;

    v = (struct block_rect *) malloc((unsigned)(nh-nl+1)*
                                     sizeof(struct block_rect));

    if (!v){
        printf("allocation error: new_block \n");
        exit(1);
    }
    v -= nl;
    for ( i = nl; i <=nh; i++) {
        v[i].previous = v[i].next = NULL;
        v[i].currentsubblock = v[i].firstsubblock = NULL;
        v[i].firstcontextblock = v[i].currentcontextblock = NULL;
        v[i].firstwhite = v[i].currentwhite = NULL;
        v[i].att = UNKNOWN;
        v[i].index = 0;
        v[i].numsubblock = 0;
        v[i].on_point = 0;
        v[i].pairight = v[i].pairleft = NULL;
        v[i].is_sorted = 0;
        v[i].cover = 0;
        v[i].group = 0;
        v[i].in_group = 0;
        v[i].density = 0.0;
    }
    return v;
}

/*****************************************************************/
/*                                                               */
/*      Chain the block                                          */
/*                                                               */
/*****************************************************************/
struct block_rect *new_rectblock(headblock)
struct block_rect *headblock;
{
    struct block_rect *n;

    n = new_block(0,0);
    if (headblock->firstsubblock == NULL)
        headblock->firstsubblock = n;
    else{
        headblock->currentsubblock->next = n;
        n->previous = headblock->currentsubblock;
    }
    headblock->currentsubblock = n;
    headblock->numsubblock++;
    return n;
}

/*****************************************************************/
/*                                                               */
/*      Allocate memory for whiteblock                           */
/*                                                               */
/*****************************************************************/
struct whiteblock *new_white(nl, nh)
int nl, nh;
{
    struct whiteblock *v;
    int i;

    v = (struct whiteblock *) malloc((unsigned)(nh-nl+1)*
                                     sizeof(struct whiteblock));
    if (!v){
        printf("allocation error: whiteblock nl = %d nh = %d \n", nl, nh);
```

【図66】

```
    exit(1);
  }
  v = v-nl;

  for ( i = nl; i <= nh; i++){
      v[i].numwhite = 0;
      v[i].white = NULL;
      v[i].previous = v[i].next = NULL;
      v[i].index = 0;
      v[i].line_with_black = 0;
      v[i].length = 1;
      v[i].pairright = v[i].pairleft = NULL;
  }
  return v;
}

/*.....*/
/*.....*/
/*.....*/
void free_pair(pair1, pair2, nl, nh)
int *pair1, *pair2;
int nl, nh;
{
  int i;

  if (pair1 == NULL) return;
  for ( i = nl; i <= nh; i++) {
      free((int *)pair1[i]);
      free((int *)pair2[i]);
  }

  free((int*) (pair1+nl));
  free((int*) (pair2+nl));
  pair1 = pair2 = NULL;
}

/*.....*/
/*.....*/
/*.....*/

void free_rectblock(nblock)
struct block_rect* nblock;
{

  free_pair(nblock->pairright, nblock->upper_y, nblock->lower_y);
  free((struct block_rect *)nblock);
  nblock = NULL;
}

/*.....*/
/*.....*/
/*.....*/

void free_white(wblock)
struct whiteblock* wblock;
{

  free_pair(wblock->pairright, wblock->pairleft, wblock->upper_y, wblock->lower_y);
  free((struct whiteblock *)wblock);
  wblock = NULL;
}

/*.....*/
/*.....*/
/*.....*/

struct data_outline *new_data_outline(nl, nh)
int nl, nh;
{
    int i;
    struct data_outline *v;

    v = (struct data_outline *) malloc((unsigned)(nh-nl+1)*
                                       sizeof(struct data_outline));
    if (!v) {
      printf("allocation error: data_outline \n");
      exit(1);
    }
    v = v-nl;
    for ( i = nl; i <=nh; i++) {
        v[i].total_left = v[i].total_right = 0;
        v[i].current_left = v[i].current_right = NULL;
        v[i].headleft = v[i].headright = NULL;
    }

    return v;
}

/*.....*/
/*.....*/
/*.....*/

struct h_outline *new_h_outline(talpack)
int talpack;
{
    struct h_outline *v;

    v = (struct h_outline *) malloc(sizeof(struct h_outline));
    if (!v) {
      printf("allocation error: h_outline \n");
      exit(1);
    }
    v->x = talpack;
    v->next = NULL;

    return v;
}

/*.....*/
/*.....*/
/*.....*/

int test_one_branch(p, x, y, newdirect, x1, y1, x2, y2)
unsigned char *p;
int         x, y, newdirect;
int         x1, y1, x2, y2;
int {
    int k, j, newx, newy;

    k = search_map[newdirect][search_map[newdirect][0]];
    for (j = k+1; j <= EAST; j++){
        newx = x+around_center[j].x;
        newy = y+around_center[j].y;
        if ((p[newy][newx] == CHANGE
             (newx >= x1)&&(newx <= x2)&&(newy >= y1)&&(newy <= y2))
                return 1;
    }
    return 0;
}

/*.....*/
/*.....*/
/*.....*/
```

【図67】

```
/*****************************************************************/
void add_x_right(x, y, ylist)
int x, y;
struct data_outline *ylist;
{
    struct data_outline *tmp;

    tmp = new_h_outline(x);
    if (ylist[y].total_right == 0){
        ylist[y].current_right = tmp;
        ylist[y].headright = tmp;
    } else {
        ylist[y].current_right->next = tmp;
        ylist[y].current_right = tmp;
    }
    ylist[y].total_right++;
}
/*****************************************************************/
/*********  Adding one element to data_outline on left side  *****/
/*****************************************************************/
void add_x_left(x, y, ylist)
int x, y;
struct data_outline *ylist;
{
    struct data_outline *tmp;

    tmp = new_h_outline(x);
    if (ylist[y].total_left == 0){
        ylist[y].current_left = tmp;
        ylist[y].headleft = tmp;
    } else {
        ylist[y].current_left->next = tmp;
        ylist[y].current_left = tmp;
    }
    ylist[y].total_left++;
}
/*****************************************************************/
/*********  Adding element to data_outline  **********************/
/*****************************************************************/
void add_y_list(x, y, location, ylist)
int x, y;
recording location;
struct data_outline *ylist;
{
    if (location == LEFT)
        add_left(x, y, ylist);
    else if (location == RIGHT)
        add_right(x, y, ylist);
    else if (location == BOTH){
            add_left(x, y, ylist);
            add_right(x, y, ylist);
        }
}
/*****************************************************************/
/*********  Sort integer array from index nl to nh  **************/
/*****************************************************************/
void bubble_sort(list, nl, nh)
int *list;
int nl, nh;
{
    int i, k, tmp, stop_sort;

    stop_sort = 0;
    for ( i = 0; i < (nh-nl); i++){
        for(k = nl, stop_sort = 0; k < nh-i; k++)
            if (list[k] > list[k+1]){
                tmp = list[k];
                list[k] = list[k+1];
                list[k+1] = tmp;
                stop_sort = 1;
            }
        if (stop_sort == 0) break;
    }
}
/*****************************************************************/
/*********  Pair the element in two integer arrays  **************/
/*****************************************************************/
void sort_x(pairright, pairleft)
int *pairright, *pairleft;
{
    int *tmppair;
    int r,l,numpair;
    int i;

    numpair = (pairright[0] <= pairleft[0])?pairright[0]:pairleft[0];
/* printf("place 1 numpair = %d \n", numpair); */
    tmppair = (int *)vector(0, 2*numpair-1);
    bubble_sort(pairright, 1, pairright[0]);
    bubble_sort(pairleft, 1, pairleft[0]);

    for (numpair = 0, r = 1, l = 1; (r <= pairright[0])&&(l <= pairleft[0]); ){
        if ((pairleft[0]<=pairleft[l])||(pairright[r]<=pairleft[l])){
            l++;
            continue;
        }
        if ((r<=pairright[0])&&(pairright[r] == pairright[r-1])){
            r++;
            continue;
        }

        if ((pairright[r] == pairleft[l])||((pairright[r]-1)<=pairleft[l])) {
            r++;
            l++;
            continue;
        }

        if (r < pairright[0]){
            if ((pairright[r]<pairleft[l])&&(pairright[r-1]<pairleft[l])){
                tmppair[numpair*2] = pairright[r];
                tmppair[numpair*2+1] = pairleft[l];
                r++; numpair++;
            } else if ((pairright[r]<pairleft[l])&&(pairright[r-1]<=pairleft[l])){
                r++;
            } else if (pairright[r]>pairleft[l]) l++;
        } else {
            if (pairright[r] < pairleft[l]){
                tmppair[numpair*2] = pairright[r];
                tmppair[numpair*2+1] = pairleft[l];
                numpair++;
                break;
            } else l++;
        }
    }
    for (i = 0; i < numpair; i++){
        pairright[i+1] = tmppair[2*i];
        pairleft[i+1] = tmppair[2*i+1];
    }
    pairright[0] = numpair;
    pairleft[0] = numpair;
}
```

【図68】

```
                                                void set_outline(x, y, nblock)
                                                int     x, y;
                                                struct whiteblock  *nblock;
                                                {
                                                    if (x > nblock->right_x )
                                                        nblock->right_x = x;
                                                    else if (x < nblock->left_x)
                                                        nblock->left_x = x;

                                                    if (y > nblock->lower_y)
                                                        nblock->lower_y = y;
                                                    else if (y < nblock->upper_y)
                                                        nblock->upper_y = y;
                                                }
                                                /*.....  set up the extreme point for the searched block  .....*/
                                                /*.....  ..... .....*/

                                                void set_white_outline(x, y, nblock)
                                                int     x, y;
                                                struct block_rect  *nblock;
                                                {

                                                    if (x > nblock->right_x )
                                                        nblock->right_x = x;
                                                    else if (x < nblock->left_x)
                                                        nblock->left_x = x;

                                                    if (y > nblock->lower_y)
                                                        nblock->lower_y = y;
                                                    else if (y < nblock->upper_y)
                                                        nblock->upper_y = y;
                                                }
                                                /*.....  .....  .....*/
                                                /*.....  Allocate int* memory  .....*/
                                                /*.....  .....  .....*/
                                                int **intpointer(nl, nh)
                                                int nl, nh;
                                                {
                                                    int **v;

                                                    if((v = ((int **) malloc((nh-nl+1)*sizeof(int *))) == NULL){
                                                        printf("allocation error; in intpointer");
                                                        exit(1);
                                                    }

                                                    v = v-nl;
                                                    return v;
                                                }

                                                /*.....  transfer outline data into pair  .....*/
                                                /*.....  .....  .....*/

                                                void pair_outline_into_pair(nblock, ylist)
                                                struct block_rect   *nblock;
                                                struct data_outline ylist[];
                                                {
                                                    int i, k;
                                                    struct h_outline *tmp1, *tmp2;

                                                    nblock->pairright = (int **)intpointer(nblock->upper_y, nblock->lower_y);
                                                    nblock->pairleft  = (int **)intpointer(nblock->upper_y, nblock->lower_y);
                                                    for (i = nblock->upper_y; i <= nblock->lower_y; i++){
```

【図69】

```
    nblock->pairright[i] = (int **)ivector(0, ylist[i].total_right);
    nblock->pairleft[i] = (int **)ivector(0, ylist[i].total_left);
    nblock->pairright[i][0] = ylist[i].total_right;
    for (tmp1 = ylist[i].headright, k = 1; tmp1 != NULL; k++){
        nblock->pairright[i][k] = tmp1->x;
        tmp2 = tmp1->next;
        free((struct h_outline *)tmp1);
        tmp1 = tmp2;
    }
    nblock->pairleft[i][0] = ylist[i].total_left;
    for (tmp1 = ylist[i].headleft, k = 1; tmp1 != NULL; k++){
        nblock->pairleft[i][k] = tmp1->x;
        tmp2 = tmp1->next;
        free((struct h_outline *)tmp1);
        tmp1 = tmp2;
    }
}

/*****************************************************************/
/*                                                               */
/*****  Transfer outline data into pair                      *****/
/*                                                               */
/*****************************************************************/

void put_hiretsuline_into_pair(nblock, ylist)
struct wiltblock  *nblock;
struct data_outline ylist[];
{
    int i, k;
    struct h_outline *tmp1, *tmp2;

    nblock->pairright = (int **)intpointer(nblock->upper_y, nblock->lower_y);
    nblock->pairleft  = (int **)intpointer(nblock->upper_y, nblock->lower_y);

    for (i = nblock->upper_y; i <= nblock->lower_y; i++){
        nblock->pairright[i] = (int *)ivector(0, ylist[i].total_right);
        nblock->pairleft[i] = (int *)ivector(0, ylist[i].total_left);
        nblock->pairright[i][0] = ylist[i].total_right;
        for (tmp1 = ylist[i].headright, k = 1; tmp1 != NULL; k++){
            nblock->pairright[i][k] = tmp1->x;
            tmp2 = tmp1->next;
            free((struct h_outline *)tmp1);
            tmp1 = tmp2;
        }
        nblock->pairleft[i][0] = ylist[i].total_left;
        for (tmp1 = ylist[i].headleft, k = 1; tmp1 != NULL; k++){
            nblock->pairleft[i][k] = tmp1->x;
            tmp2 = tmp1->next;
            free((struct h_outline *)tmp1);
            tmp1 = tmp2;
        }
    }
}

/*****************************************************************/
/*                                                               */
/*****  Convert block into head block                        *****/
/*                                                               */
/*****************************************************************/

void converse_headblock(first)
struct block_rect *first;
{
    struct block_rect* n;

    if ((first->firstsubblock != NULL) return;
    n = new_block(0,0);

    n->index = 1;
    n->upper_y = first->upper_y;
    n->lower_y = first->lower_y;
    n->left_x = first->left_x;
    n->right_x = first->right_x;
    n->x = (first->x);
    n->y = first->y;
    n->width = first->width;
    n->length = first->length;
    n->con_point = first->con_point;
    n->att = first->att;
    n->density = first->density;
    n->is_ported = first->is_ported;
    n->pairright = first->pairright;
    n->pairleft = first->pairleft;
    first->pairright = NULL;
    first->pairleft = NULL;
    n->power = ONTOINA;

    first->numsubblock = 1;
    first->firstsubblock = n;
    first->currentcontextblock = NULL;
    first->firstnoncontextblock = NULL;
/*printf("get out of conversion");*/
}

/*****************************************************************/
/*****  Add one block under the specified block by the y location  *****/
/*****************************************************************/

void add_rectblock(text, first, head)
int
struct block_rect *first;
struct block_rect *head;
{
    struct block_rect* tmpb;

    if ((first->upper_y < head->upper_y)){
        head->upper_y = first->upper_y;
        head->length = head->lower_y-head->upper_y+1;
    }
    if ((first->lower_y > head->lower_y)){
        head->lower_y = first->lower_y;
        head->length = head->lower_y-head->upper_y+1;
    }
    if ((first->left_x < head->left_x)){
        head->left_x = first->left_x;
        head->width = head->right_x-head->left_x+1;
    }
    if ((first->right_x > head->right_x)){
        head->right_x = first->right_x;
        head->width = head->right_x-head->left_x+1;
    }

    if (text != TEXT) {
        if (head->firstnoncontextblock == NULL){
            head->firstnoncontextblock = first;
            first->previous = first->next = NULL;
            head->currentnoncontextblock = first;
        }
        else {
            for (tmpb = head->currentnoncontextblock; tmpb != tmpb->previous;){
                if (tmpb == NULL) {
                    first->previous = NULL;
                    first->next = head->firstnoncontextblock;
                    head->firstnoncontextblock->previous = first;
```

【図70】

```
        head->firstnextextblock = first;
        break;
      }
      else if (first->upper_y < tmpb->upper_y) continue;
           else {
                first->previous = tmpb;
                first->next = tmpb->next;
                tmpb->next = first;
                if (first->next != NULL)
                  first->next->previous = first;
                else
                  head->currentnextextblock = first;
                break;
              }
    }
  }
  if (head->firstsubblock == NULL) {
      head->firstsubblock = first;
      first->previous = first->next = NULL;
      head->currentsubblock = first;
    }
  else {
    for (tmpb = head->currentsubblock; ; tmpb = tmpb->previous){
      if (tmpb == NULL) {
                first->previous = NULL;
                first->next = head->firstsubblock;
                head->firstsubblock->previous
                                    = first;
                head->firstsubblock = first;
                break;
              }
      else if (first->upper_y < tmpb->upper_y) continue;
           else {
                first->previous = tmpb;
                first->next = tmpb->next;
                tmpb->next = first;
                if (first->next != NULL)
                  first->next->previous = first;
                else
                  head->currentsubblock = first;
                break;
              }
    }
  }
}

/*****************************************************************/
/*                                                               */
/*      Remove white block                                       */
/*                                                               */
/*****************************************************************/

void remove_white(nblock, wblock)
struct block_rect *nblock;
struct whiteblock *wblock;
{
  if (nblock->firstwhite == wblock){
      nblock->firstwhite = wblock->next;
      if (wblock->next != NULL) wblock->next->previous = NULL;
      else nblock->currentwhite = NULL;
    }
  else if (nblock->currentwhite == wblock){
      nblock->currentwhite = wblock->previous;
      wblock->previous->next = NULL;
    } else {
        wblock->previous->next = wblock->next;
        wblock->next->previous = wblock->previous;
      }
}

/*****************************************************************/
/*                                                               */
/*      Remove text block                                        */
/*                                                               */
/*****************************************************************/

void remove_textblock(first, head)
struct block_rect *first, *head;
{
  if (head->firstsubblock == first){
      head->firstsubblock = first->next;
      if (first->next != NULL) first->next->previous = NULL;
        else head->currentsubblock = NULL;
    }
  else if (head->currentsubblock == first){
      head->currentsubblock = first->previous;
        first->previous->next = NULL;
      } else {
          first->previous->next = first->next;
          first->next->previous = first->previous;
        }
}

/*****************************************************************/
/*                                                               */
/*      Remove context block                                     */
/*                                                               */
/*****************************************************************/

void remove_contextblock(first, head)
struct block_rect *first, *head;
{
  if (head->firstcontextblock == first){
      head->firstcontextblock = first->next;
      if (first->next != NULL) first->next->previous = NULL;
        else head->currentcontextblock = NULL;
    }
  else if (head->currentcontextblock == first){
      head->currentcontextblock = first->previous;
        first->previous->next = NULL;
      } else {
          first->previous->next = first->next;
          first->next->previous = first->previous;
        }
}

/*****************************************************************/
/*                                                               */
/*      Remove nontext block                                     */
/*                                                               */
/*****************************************************************/

void remove_nontextblock(first, head)
struct block_rect *first, *head;
{
  if (head->firstnontextblock == first){
      head->firstnontextblock = first->next;
      if (first->next != NULL) first->next->previous = NULL;
        else head->currentnontextblock = NULL;
    }
  else if (head->currentnontextblock == first){
      head->currentnontextblock = first->previous;
        first->previous->next = NULL;
      } else {
          first->previous->next = first->next;
          first->next->previous = first->previous;
        }
}

/*****************************************************************/
/*                                                               */
/*      Transfer block from one head to another head block       */
/*                                                               */
/*****************************************************************/

void transfer_block(fcontext, tcontext, first, originalhead, head)
int fcontext, tcontext;
struct block_rect *first, *originalhead, *head;
{
/*printf("go into transfer block\n");*/
    if (fcontext == TEXT
        remove_textblock(first, originalhead);
    else
        remove_contextblock(first, originalhead);

  add_textblock(tcontext, first, head);
  if (originalhead != head) {
    originalhead->numsubblock--;
```

[図71]

```
        head->numsubblock++;
    }
/*printf("get out of  transfer block\n");*/
}


/*************************************************************************/
/********  Rearrange the nested relation among the block          ********/
/*************************************************************************/

void rearrange_rectblock(headblock)
struct block_rect *headblock;
{
    struct block_rect *tmp, *tmp1, *tmpb, *tmpbb, *b;

/*  printf("get into rearrange_rectblock\n"); */

    tmp = headblock->firstsubblock;
    for ( ; tmp != NULL; tmp = tmpbb){
      tmpbb = tmp->next;
      for ( tmp1 = tmp->next; tmp1 != NULL; tmp1 = tmp1->next){
        if (tmp->lower_y < tmp1->upper_y)
            break;
        else{
            if ((tmp->lower_y>=tmp1->lower_y)&&
                (tmp->left_x<=tmp1->left_x)&&
                (tmp->right_x >= tmp1->right_x)){
                converto_headblock(tmp);
                tmp->cover = COVER;
                tmpb = tmp1->previous;
                tmp1->cover = COVERED;
                transfer_block(TEXT, TEXT, tmp1, headblock, tmp);
                if (tmpbb == tmp1) tmpbb = tmp->next;

                tmp1 = tmpb;
            } else if ((tmp1->lower_y >= tmp->lower_y)&&
                       (tmp1->left_x <= tmp->left_x)&&
                       (tmp1->right_x >= tmp->right_x)&&
                       (tmp->upper_y >= tmp1->upper_y)){
                   converto_headblock(tmp1);
                   tmp1->cover = COVER;
                   if (tmp->cover == COVER){
                      for (b = tmp->firstsubblock; b != NULL;
                                                   b = b->next){
                         transfer_block(TEXT, TEXT, b, tmp, tmp1);
                         b->cover = COVERED;
                      }
                      remove_textblock(tmp, headblock);
                   } else {
                          tmp->cover = COVERED;
                          transfer_block(TEXT, TEXT, tmp,
                                                headblock, tmp1);
                   }

                   break;
                }
            }
        }
    }
/*  printf("get out of  rearrange_rectblock\n"); */
}

/*************************************************************************/
/********  search 8-connected "one" component                    ********/
/*************************************************************************/

void search_one(p, startx,  starty, x1,  y1,  x2,  y2, nblock)
unsigned char       **p;
int                 startx, starty, x1, y1, x2, y2;
struct block_rect   *nblock;
{
    int oldx, oldy, newx, newy;
    int olddirect, newdirect;
    recording location;
    struct data_outline *ylist_on;
    int i, j;

/* printf("get into search one x = %d y = %d\n", startx, starty);  */

    ylist_on =(struct data_outline*)new_data_outline(y1, y2);

    nblock->x = startx;
    nblock->y = starty;
    nblock->left_x  = nblock->right_x = startx;
    nblock->upper_y = nblock->lower_y = starty;

    if (test_one_branch(p, startx, starty, EAST, x1, y1, x2, y2)==0){
       nblock->width = 1; nblock->length = 1;  nblock->density = 1.0;
       nblock->on_point = 1;
       p[starty][startx] = LEFTRIGHT_ONE;
       free((struct data_outline *)(ylist_on+y1));
       return;
    }
/*
for ( i = starty-5; i <= starty+20; i++){
   printf("%3d ", i);
   for (j = startx-10; j <= startx+20; j++)
      printf("%2x ", p[i][j]);
   printf("\n");
}
*/

    if (p[starty][startx+1]&0x01){
       add_y_list(startx, starty, RIGHT, ylist_on);
       p[starty][startx] = RIGHT_ONE;
    }
    else{
         add_y_list(startx, starty, XL, ylist_on);
         p[starty][startx] = LEFTRIGHT_ONE;
       }

    olddirect = SOUTH;
    right_most_one(p, olddirect, startx, starty, x1, y1, x2, y2,
                   &newdirect, &newx, &newy, &location);
    set_outline(newx, newy, nblock);
    oldx = newx; oldy = newy;
    olddirect = newdirect;
    for ( ; ; ){
       do {
            right_most_one(p, olddirect, oldx, oldy, x1, y1, x2, y2,
                           &newdirect, &newx, &newy, &location);

            if (location != SKIP)
                    add_y_list(oldx, oldy, location, ylist_on);

            if ((location == SKIP)&&(p[oldy][oldx]==ONE))
                          p[oldy][oldx] = LEFTRIGHT_ONE;
            if (location == LEFT)
                 p[oldy][oldx] = LEFT_ONE;

            if (location == RIGHT) {
               if (p[oldy][oldx] != LEFT_ONE)
                    p[oldy][oldx] = RIGHT_ONE;
            }
```

【図72】

```
    if (location == RL) p[oldy][oldx] = LEFTRIGHT_ONE;

    direct_stack[olddirect][newdirect]--;  */
/*  printf("%d %s \n", oldx, oldy, locatext[location]);*/
        set_outline(newx, newy, nblock);
        if (p[newy][newx] == ONE) nblock->on_point++;
        oldx = newx;
        oldy = newy;
        olddirect = newdirect;
    } while (!(newx != startx)||(newy != starty));
    if (test_one_branch(p, newx, newy, newdirect, x1, y1, x2, y2) == 0)
        break;
}
nblock->on_point++;

put_outline_into_pair(nblock, ylist_on);
free((struct data_outline*)ylist_on->ll);
nblock->oldx = nblock->right_p->nblock->left_x+1;
nblock->length = nblock->lower_y - nblock->upper_y+1;
return;
}

/*.....*/
/*.....  Search the existence of surrounding 'zero'  .....*/
/*.....*/

int test_zero_branch(p, x, y, newdirect)
unsigned char **p;
int           x, y, newdirect;
{
    int k, j, newx, newy;

    k = search_map[newdirect][search_map[newdirect][0]];
    for (j = k+1; j <= EAST; j++){
        newx = k+around_center[j].x;
        newy = y+around_center[j].y;
        if (p[newy][newx] == ZERO){
            if ((j == EAST)||(j == SOUTH)) return 1;
            if (j == SE){
                if ((p[newy-1][newx]&0x01)&&(p[newy][newx-1]&0x01))
                    continue;
                else
                    return 1;
            }
            if (j == SW){
                if ((p[newy-1][newx]&0x01)&&(p[newy][newx+1]&0x01))
                    continue;
                else
                    return 1;
            }
            if (j == NE){
                if ((p[newy][newx-1]&0x01)&&(p[newy+1][newx]&0x01))
                    continue;
                else
                    return 1;
            }
            if (j == NW){
                if ((p[newy+1][newx]&0x01)&&(p[newy][newx+1]&0x01))
                    continue;
                else
                    return 1;
            }
        }
    }
    return 0;
}

/*.....*/
/*.....  Search 4-connected 'zero' component  .....*/
/*.....*/

int search_zero(p, startx, starty, nblock)
unsigned char  **p, startx, starty;
int            startx, starty;
struct block_rect   *nblock;
{
    int         oldx, oldy, newx, newy;
    int         olddirect, newdirect, white;
    recording   location;
    struct data_outline *ylist_off;
    struct whiteblock *tapwhite;

/*  printf("get into search zero x = %d y = %d \n", startx, starty);  */

    white = 0;
    if(test_zero_branch(p, startx, starty, EAST) == 0){
        p[starty][startx] = XIGHLEFT_ZERO;
        return white;
    }
    ylist_off = (struct data_outline*)new_data_outline(nblock->upper_y,
                                                       nblock->lower_y);
    tapwhite = new_white(0,0);
    tapwhite->x = startx;
```

【図73】

```
tmpwhite->y = starty;
tmpwhite->upper_y = starty;
tmpwhite->lower_y = starty;
tmpwhite->left_x = startx;
tmpwhite->right_x = startx;

if (plaistarty][startx]!=LEFT_ZERO){
    add_y_list(startx, starty, RU, ylist_off);
    p[starty][startx] = RIGHTLEFT_ZERO;
}
else{
    add_y_list(startx, starty, RIGHT, ylist_off);
    p[starty][startx] = RIGHT_ZERO;
}
olddirect = SOUTH;
right_most_zero(p, olddirect, startx, starty,
                &newdirect, &newx, &newy, &location);
set_white_outline(newx, newy, tmpwhite);
oldx = newx; oldy = newy;
olddirect = newdirect;

for ( ; ; ){
  do {  right_most_zero(p, olddirect, oldx, oldy,
                        &newdirect, &newx, &newy, &location);
        if (location != SKIP)
            add_y_list(oldx, oldy, location, ylist_off);

        if ((location == SKIP)&&(p[oldy][oldx]==ZERO))
            p[oldy][oldx] = RIGHTLEFT_ZERO;
        if (location == LEFT)
            p[oldy][oldx] = LEFT_ZERO;
        else if (location == RIGHT)
            if (p[oldy][oldx] != LEFT_ZERO)
                p[oldy][oldx] = RIGHT_ZERO;
        if (location == RL) p[oldy][oldx] = RIGHTLEFT_ZERO;
        set_white_outline(newx, newy, tmpwhite);
        oldx = newx;
        oldy = newy;
        olddirect = newdirect;
    } while ((newx != startx)||(newy != starty));
    if(test_zero_branch(p, newx, newy, newdirect) == 0)
        break;
}

tmpwhite->length = tmpwhite->lower_y - tmpwhite->upper_y+1;
tmpwhite->width = tmpwhite->right_x - tmpwhite->left_x+1;
if (((tmpwhite->length > 3)||(tmpwhite->width > 3))&&
    (tmpwhite->length >= 3)&&(tmpwhite->width >=3)){
    put_whiteoutline_into_pair(tmpwhite, ylist_off);
    if (nblock->firstwhite == NULL)
      nblock->firstwhite = tmpwhite;
    else{
        tmpwhite->previous = nblock->currentwhite;
        nblock->currentwhite->next = tmpwhite;
    }
    nblock->currentwhite = tmpwhite;
    nblock->currentwhite->next_upper_y = nblock->currentwhite->upper_y;
    nblock->currentwhite->next_lower_y = nblock->currentwhite->lower_y;
    nblock->currentwhite->next_left_x = nblock->currentwhite->left_x;
    nblock->currentwhite->next_right_x = nblock->currentwhite->right_x;
    white = 1;
}

free((struct data_outline *)ylist_of(nblock->upper_y));
return white;
}

/*****************************************************************************/
/*********  search block in white                                    *********/
/*****************************************************************************/

int block_in_white(nblock, p, tmpwhite)
struct block_rect *nblock;
unsigned char **p;
struct whiteblock *tmpwhite;
{
    int i, j, k, getin, block_no, keep;
    struct block_rect *curr_block;

/*
printf("get into block_in_white\n");
*/
    for ( i = tmpwhite->upper_y; i <= tmpwhite->lower_y; i++){
/*
printf("i = %d \n", i);
        for (j = 1; j <= tmpwhite->pairright[i][0]; j++)
            printf("%d ", tmpwhite->pairright[i][j]);
        printf("\n");

        for (j = 1; j <= tmpwhite->pairleft[i][0]; j++)
            printf("%d ", tmpwhite->pairleft[i][j]);
        printf("\n");
*/
        next_x(tmpwhite->pairright[i], tmpwhite->pairleft[i]);
/*
        printf("pair = %d \n", tmpwhite->pairright[i][0]);
        for (j = 1; j <= tmpwhite->pairright[i][0]; j++)
            printf("%d ", tmpwhite->pairright[i][j]);
        printf("..........");
        for (j = 1; j <= tmpwhite->pairleft[i][0]; j++)
            printf("%d ", tmpwhite->pairleft[i][j]);
        printf("\n");
*/
    }

    for ( i = tmpwhite->upper_y; block_no = 0, i <= tmpwhite->lower_y; i++){
    for ( j = 1; j <= tmpwhite->pairright[i][0]; j++){
/*
printf("i = %d left = %d \n", i, tmpwhite->pairright[i][j]);
*/
        for (k =(tmpwhite->pairright[i][j])-1, getin = 0;
                k < tmpwhite->pairleft[i][j]; k++ ){
            if ((getin == 0) && (p[i][k] == RIGHT_ONE)) {
                getin = 1;
            } else if ((getin == 1)&&(p[i][k] == LEFT_ONE)){
                getin = 0;
            } else if ((getin == 0)&&(p[i][k] == ONE)){
                curr_block = new_rectblock(nblock);
                search_one(p, k, i, tmpwhite->left_x,
                           tmpwhite->upper_y, tmpwhite->right_x,
                           tmpwhite->lower_y, curr_block);
/*
printf("after search_one\n");
*/
                curr_block->index = ++block_no;
                curr_block->in_group = nblock->group-1;
/*
printf("index = %d in_group = %d\n",curr_block->index, curr_block->in_group);
*/
                for ( ; (k<tmpwhite->pairleft[i][j])&&
                        (p[i][k]&(block01)); k++);
```

【図74】

```
                    tmpwhite->index = nblock->group;
                    tmpwhite = tmpwhite->next;
                }
                else {
                    tmp = tmpwhite->next;
                    remove_white(nblock, tmpwhite);
                    free_white(tmpwhite);
                    tmpwhite = tmp;
                }
            }

/*  for (tmp1= nblock->firstsubblock; tmp1 != NULL; tmp1 = tmp1->next)
        printf("index = %d, in_group = %d\n", tmp1->index, tmp1->in_group);
*/
        }

/*****************************************************************************/
/*********  Search block in the range specified                      *********/
/*****************************************************************************/
void block_in_range(p, x1, y1, x2, y2, headblock)
unsigned char   *p;
int             x1, y1, x2, y2;
struct block_rect   *headblock;
{
    int
    struct block_rect *n;

    for (i = y1; i <= y2; i++)
        for (j = x1, getin = 0; j <= x2; ){
            if (getin == 0) {  if (p[i][j] == ZERO||p[i][j] == LEFTRIGHT_ONE)||
                p[i][j] == LEFT_ONE){
                j++;
                continue;
            }
            if (p[i][j] == ONE){
                n = new_rectblock(headblock);
                n->index = headblock->numsubblock;
                search_one(p, j, i,  x1, y1, x2, y2, n);

/*
printf("index = %d wid = %d x1 = %d y1 = %d i = %d j = %d \n",
    n->index, n->length, n->width, n->left_x,  n->upper_y, i, j);
*/
                for ( ; (j <= x2)&&(p[i][j]!=0x01); j++);
                continue;
            }
            if (p[i][j] == RIGHT_ONE){
                getin = 1;
                j++;
                continue;
            }
        }
}

/*****************************************************************************/
/*********  search block in block                                    *********/
/*****************************************************************************/
void search_block_in_block(nblock, p)
struct block_rect *nblock;
unsigned char *p;
{
    struct whiteblock *tmpwhite, *tmp;
    int   i, block_no, keep;
    struct block_rect *tmp1;

    nblock->group = 0;
    for ( tmpwhite = nblock->firstwhite; tmpwhite != NULL; )
        {
/*
printf("tmpwhite index = %d numsub =%d\n", tmpwhite->index, tmpwhite->numsubwhite);
*/
        keep = 0;
        if (tmpwhite->numsubwhite != 0){
            for (i = 0; i < tmpwhite->numsubwhite; i++){
/*
printf("i = %d \n", i);
*/
                if (block_in_white(nblock, p, &(tmpwhite->subwhite[i]))){
                    keep = 1;
                    if (tmpwhite->subwhite[i].line_with_black)
                        tmp1 = tmpwhite->line_with_black;
                    else
                        tmpwhite->subwhite[i].line_with_black =
                            tmpwhite->subwhite[i].line_with_black;
                }
            }
            if (keep == 0) {
                for (i = 0; i < tmpwhite->numsubwhite; i++){
                    free_pair(tmpwhite->subwhite[i].pairright,
                              tmpwhite->subwhite[i].pairleft,
                              tmpwhite->subwhite[i].upper_y,
                              tmpwhite->subwhite[i].lower_y);
                }
                tmpwhite->pairright = tmpwhite->pairleft = NULL;
                free((struct whiteblock *)tmpwhite->subwhite);
            }
            tmpwhite->numsubwhite = 0;
        }
        if (block_in_white(nblock, p, tmpwhite))
            keep = 1;

        if (keep) {
            nblock->group++;
```

【図75】

```
    }
/* printf("get out of block in range\n"); */
}

/*****************************************************************/
/*******                                                 *******/
/******* count the density inside the block              *******/
/*******                                                 *******/
/*****************************************************************/
void sort_for_block(block)
struct block_rect *block;
{
    int i;

    if (block->if_sorted == 0){
        for (i = block->upper_y; i<= block->lower_y; i++)
            sort_1(block->pairright[i], block->pairleft[i]);
        block->if_sorted = 1;
    }
}

/*****************************************************************/
/*******                                                 *******/
/*******  filled area  for the group in a big block      *******/
/*******                                                 *******/
/*****************************************************************/
void fill_white_in_rect(nblock, area_filled)
struct block_rect *nblock;
float *area_filled;
{
    struct whiteblock *tmpwhite;
    int   top, bottom, left, right;
    int   x, y, pair;
    unsigned char *square;

    top = nblock->lower_y;
    bottom = nblock->upper_y;
    left = nblock->right_x;
    right = nblock->left_x;
    for (tmpwhite = nblock->firstwhite; tmpwhite != NULL;
                tmpwhite = tmpwhite->next){
        if (tmpwhite->left_x < left) left = tmpwhite->left_x;
        if (tmpwhite->right_x > right) right = tmpwhite->right_x;
        if (tmpwhite->upper_y < top) top = tmpwhite->upper_y;
        if (tmpwhite->lower_y > bottom) bottom = tmpwhite->lower_y;
    }
/*
    printf("top = %d lower = %d left=%d right=%d \n", top, bottom, left, right);
*/
    square = cmatrix(top, bottom, left, right);

    for (tmpwhite = nblock->firstwhite; tmpwhite != NULL;
                tmpwhite = tmpwhite->next){
        for (y = tmpwhite->upper_y; y <= tmpwhite->lower_y; y++){
            for (x = tmpwhite->left_x; x <= tmpwhite->right_x; x++)
                square[y][x] = 1;
        }
    }

    for (pair = 0; y = top; y <= bottom; y++)
        for ( x = left; x <= right; x++)
            if (square[y][x] == 1)
                pair++;

    *area_filled = (float)pair/(float)((bottom-top+1)*(right-left+1));
    free_cmatrix(square, top, bottom, left, right);
}

/*****************************************************************/
/*******                                                 *******/
/******* count continuous one and zero                   *******/
/*******                                                 *******/
/*****************************************************************/
void cont_one_zero(nblock, p,  avg_one, std_one, avg_zero, std_zero)
struct block_rect  *nblock;
unsigned char **p;
float *avg_one, *std_one, *avg_zero, *std_zero;
{

    int i, j, k;
    int one_count, zero_count;
    int *one, *zero;
    int no_one, no_zero;

/*
printf("place 1\n");
*/

    one = ivector(0, nblock->length*nblock->width-1);
    zero = ivector(0, nblock->length*nblock->width-1);
    no_one = 0;     no_zero = 0;

    for (i = nblock->upper_y; i <= nblock->lower_y; i++){
        one_count = 0;
        zero_count = 0;
        for (j = 1; j <= nblock->pairright[i][0]; j++){
            for (k = nblock->pairleft[i][j]; k<=
                    nblock->pairright[i][j]; k++){
                if (p[i][k]==0) {
                    one_count++;
                    if (zero_count != 0){
                        zero[no_zero++] = zero_count;
                        zero_count  = 0;
                    }
                } else {
                    zero_count++;
                    if (one_count != 0) {
                        one[no_one++] = one_count;
                        one_count = 0;
                    }
                }
            }
        }
        if (one_count != 0) {
            one[no_one++] = one_count;
            one_count = 0;
        }
    }
/*
printf("no_one = %d no_zero = %d\n", no_one, no_zero);
*/
    if (no_one > 0) *avg_one = *std_one = 0;
    if (no_zero > 0) *avg_zero = *std_zero = 0;
    if (no_one > 0) avg_std(one, no_one, avg_one, std_one);
    if (no_zero > 0) avg_std(zero, no_zero, avg_zero, std_zero);

/*
printf("avg_one=%f std_one=%f avg_zero=%f std_zero=%f  \n",
    *avg_one, *std_one, *avg_zero, *std_zero);
*/

    free_ivector(one, 0, nblock->length*nblock->width-1);
    free_ivector(zero, 0,  nblock->length*nblock->width-1);
}
```

【図76】

```
void cost_over_and_cmszero(nblock, p,  avg_one, std_one, avg_zero, std_zero, no_over)struct block_rect *nblock;
unsigned char *p;
float *avg_one, *std_one, *avg_zero, *std_zero;
int   *no_over;
{
    int i, j, k;
    int one_count, zero_count;
    int *one, *zero;
    int no_one, no_zero;

/*
printf("place 4\n");
*/
    one = ivector(0, nblock->length*nblock->width-1);
    zero = ivector(0, nblock->length*nblock->width-1);
    no_one = 0;
    no_zero = 0;
    *no_over = 0;
    for (i = nblock->upper_y; i <= nblock->lower_y; i++){
        one_count = 0;
        zero_count = 0;
        for (j = 1; j <= nblock->pairright[i][0]; j++){
            for (k = nblock->pairright[i][j]; k <= nblock->pairleft[i][j]; k++){
                if (p[i][k]&0x01) {
                    one_count++;
                    if (zero_count != 0){
                        zero[no_zero++] = zero_count;
                        zero_count = 0;
                    }
                } else {
                    zero_count++;
                    if (one_count != 0) {
                        one[no_one++] = one_count;
                        if (one_count > MAXFONT/SCALE_RATIO) (*no_over)++;
                        one_count = 0;
                    }
                }
            }
        }
        if (one_count != 0) {
            one[no_one++] = one_count;
            if (one_count > MAXFONT/SCALE_RATIO) (*no_over)++;
            one_count = 0;
        }
    }
/*
printf("no_one = %d no_zero = %d\n", no_one, no_zero);
*/
    *avg_one = *avg_zero = *std_zero = 0;
    if (no_one > 0) avg_std(one, no_one, avg_one, std_one);
    if (no_zero > 0) avg_std(zero, no_zero, avg_zero, std_zero);
/*
printf("avg_one=%f std_one=%f avg_zero=%f std_zero=%f no_over = %d \n",
    *avg_one, *std_one, *avg_zero, *std_zero, *no_over);
*/
    free_ivector(one, 0, nblock->length*nblock->width-1);
    free_ivector(zero, 0, nblock->length*nblock->width-1);
}

/*****************************************************************/
/*******  sort block                                             */
/*****************************************************************/
void sort_block(headblock)
{
    struct block_rect** block_y;
    struct block_rect* tmp;
    int i, j, num;
    int change;

    for (tmp = headblock->firstsubblock, num = 0; tmp != NULL;
                                        tmp = tmp->next, num++ );

    if (num == 0) return;

    if ((block_y = (struct block_rect**)
                malloc(num*sizeof(struct block_rect*))) == NULL){
        printf("allocatio error: in sort_block\n");
        exit(1);
    }

    for (tmp = headblock->firstsubblock, i = 0; tmp != NULL; tmp = tmp->next)
        block_y[i++] = tmp;

    for ( i = 0; i < num-1 ; i++){
        for (j = 0, change = 0; j < num-1-i; j++)
            if (block_y[j]->upper_y > block_y[j+1]->upper_y){
                change = 1;
                tmp = block_y[j+1];
                block_y[j+1] = block_y[j];
                block_y[j] = tmp;
            }
        if (change == 0) break;
    }
/*
printf("in sort_block i = %d \n", i);
*/
    headblock->firstsubblock = block_y[0];
    block_y[0]->next = block_y[1];
    for (i = 1; i < num-1; i++){
        block_y[i]->previous = block_y[i-1];
        block_y[i]->next = block_y[i+1];
    }
    block_y[num-1]->previous = block_y[num-2];
    block_y[num-1]->next = NULL;
    headblock->currentsubblock = block_y[num-1];
    free((struct block_rect **)block_y);
}

/*****************************************************************/
/*******  discrimination between line_drawing and grey-scale picture  *******/
/*****************************************************************/
void pic_grey_discriminate(nblock, p, black_per, black_density)
struct block_rect *nblock;
unsigned char **p;
float black_per, black_density;
{
    float avg_one, std_one, avg_zero, std_zero;

/*
printf("go into pic_grey_dis\n");
*/
    cont_one_zero(nblock, p, &avg_one, &std_one, &avg_zero, &std_zero);

    if (avg_zero == 0.0){
        if (black_per > 0.1 )
            nblock->bit |= PICTURE;
        else
```

【図77】

```
    nblock->att |= PICTURE;
  } else if (avg_zero > avg_one)
      nblock->att |= PICTURE;
    else if (black_density > 0.5)
        nblock->att |= HPICTURE;
      else if (black_par > 0.1)
          nblock->att |= HPICTURE;
        else
          nblock->att |= PICTURE;

  /*
    if ((black_par < 0.3)||(avg_zero > avg_one))
        nblock->att |= PICTURE;
    else
        nblock->att |= HPICTURE;
  */

  /*
    if (((avg_one > avg_zero)&&(black_par >= 0.5))||((avg_one > avg_zero*2.0)
        nblock->att |= HPICTURE;
    else nblock->att |= PICTURE;
  */

    if (black_par >= 0.5) nblock->att |= HPICTURE;
    else nblock->att |= PICTURE;
  /*
  printf("att = %x\n", nblock->att);
  printf("get out of  pic_grey_div\n");
  */
}

/*.....*/
/*.....Search block in the block specified                    .....*/
/*.....                                                       .....*/

void block_in_block(p, nblock, first)
unsigned char *p;
struct block_rect   *nblock, *first;
{
      int   i, j, k, getin, n, total_one, total_zero, no_over;
      float avg_one, std_one, avg_zero, std_zero;
      float area_filled, black_par, black_density;
      struct block_rect *tmp;

  /*
  printf("\n\n(*************************************)\n");
  printf("get into block \n x = %d y1 = %d y2 = %d  i . %d j . %d\n",
          nblock->left_x, nblock->upperly, nblock->lower_y,
          nblock->length, nblock->width);
  */

    if (nblock->length <= 2){
        nblock->att |= HSOLID;
  /*
  printf("att = %x\n", nblock->att);
  printf("\n(*************************************)\n\n");
  */
      return;
    }
    if (nblock->width <= 2){
        nblock->att |= VSOLID;
  /*
  printf("att = %x\n", nblock->att);
  printf("\n(*************************************)\n\n");
  */
      return;
    }
    sort_for_block(nblock);

    total_one = 0;
    total_zero = 0;
    for (i = nblock->upperly; i <= nblock->lower_y; i++){
        for (j = 3; j <= nblock->pairright[i][0] / 3+4; ){
            for (k = nblock->pairright[i][j]; getin = 0;
                 k <= nblock->pairleft[i][j]; k++){

      if (p[i][k]&0x0020) {
            total_zero++;
      } else {
            total_one++;
            if ((p[i][k] == RIGHT_ZERO)&&(getin == 0)){
                getin = 1;
            } else if ((p[i][k] == LEFT_ZERO)&&(getin ==1)){
                getin = 0;
            } else if ((p[i][k] == ZERO)&&(getin == 0)){
                if (search_zero(p, k, i, nblock)){
                    nblock->group++;
                    nblock->currentwhite->index =
                                        nblock->group;
                }
                for (k++; (k < nblock->pairleft[i][j])
                          &&((p[i][k]&0x0020)); k++);
                k--;
                continue;
            }
        }
    }
    black_density = (float)total_one/((float)total_one+total_zero);
    black_par = ((float)total_one/((float)nblock->length*nblock->width));
  /*
  printf("group = %d black_par = %f black_density=%f \n", nblock->group, black_par,
                                        black_density);
  */

    if (nblock->group >= TABLE_GROUP){
        fill_white_info_rect(nblock, &area_filled);
  /*
  printf("area_filled = %f \n", area_filled);
  */
        if (area_filled >= MINIMUM_SPACE)
          if (!(table(nblock)) nblock->att |= TABLE;
        if (!(nblock->att&TABLE))
          pic_grey_discriminate(nblock, p, black_par, black_density);
    } else if ((!(nblock->att&PAGE))||((black_par >= 0.5)){
          if (nblock->length > MAXFONT/SCALE_RATIO)
            pic_grey_discriminate(nblock, p, black_par, black_density);
          else if (nblock->width > MAXFONT/SCALE_RATIO){
            cont_over_and_nozero(nblock, p, &avg_one, &std_one,
                                 &avg_zero, &std_zero, &no_over);

            if (no_over > 0)
            {
              if (((avg_one > avg_zero)&&(black_par >= 0.5))||
                  (avg_one > avg_zero*2.0))
                nblock->att |= HPICTURE;
              else nblock->att |= PICTURE;
            }
          }

    if ((nblock->att != UNKNOWN)&&(nblock->att!=UNKNOWN))
      nblock->att = UNKNOWN;
```

【図78】

```
    if ((nblock->att == FRAME)||(nblock->att&TABLE)){

/*
printf("before search block in block \n");
*/
        search_block_in_block(nblock, p);
/*
printf("after search block in block \n");
*/
        if ((nblock->firstsubblock != NULL)&&(nblock->group >= 2))
            sort_block(nblock);

/*
for (tmp = nblock->firstsubblock; tmp != NULL; tmp = tmp->next)
    printf("index = %d is_group = %d \n", tmp->index, tmp->is_group);
*/
        rearrange_rectblock(nblock);
    }

/*
for (tmp = nblock->firstsubblock; tmp != NULL; tmp = tmp->next)
    printf("index = %d is_group = %d \n", tmp->index, tmp->is_group);
*/
    if((nblock->pairright != NULL) free_pair(nblock->pairright, nblock->pairleft);printf("get out of _text_discriminate \n"); */
                    nblock->upper_y, nblock->lower_y));
}

/*
printf("att = %x\n", nblock->att);
printf("\n**********.....**********\n\n");
*/
    return;
/* printf("get out of  block in block \n");*/
}

/*****************************************************************/
/********* Delete some block from the head block *****************/
/*****************************************************************/
void delete_rectblock(first, headblock)
struct block_rect *first, *headblock;
{
    remove_textblock(first, headblock);
    headblock->numsubblock--;
    free_rectblock(first);
}

/*****************************************************************/
/********* first time text discrimination ************************/
/*****************************************************************/
int _text_discriminate(headblock)
struct block_rect *headblock;
{
    int     num = 0;
    long    templength = 0;
    float   temp_length;
    struct block_rect *tmp, *tmpb;

/*printf("get into _text_discriminate \n");*/
    temp_length = (float)130.0/SCALE_RATIO;
    for (tmp = headblock->firstsubblock; tmp != NULL; ){
        if (((float)tmp->length > temp_length)
            || (tmp->width > headblock->width/WIDTH_RATIO)){
            tmp->att = UNKNOWN;
            if (tmp->cover == COVER) {
                for (tmpb = tmp->firstsubblock; tmpb != NULL; tmpb = tmpb->next)
                    if (tmpb->cover == ORIGINAL) {
                        transfer_block(TEXT, NONTEXT, tmp, tmpb);
                        break;
                    }
            }
            tmpb = tmp->next;
            transfer_block(TEXT, NONTEXT, tmp, headblock, headblock);
            tmp = tmpb;
        }
        else {
            templength += (long)tmp->length;
            num++;
            tmp->att = TEXT;
            tmp = tmp->next;
        }
    }
    if (num > 0)
        headblock->avg_height = (float)templength/(float)num;
    else
        headblock->avg_height = 0.0;
    return num;
}

/*****************************************************************/
/********* split a block *****************************************/
/*****************************************************************/
void block_split(head, tmp, gap1, gap2)
struct block_rect *tmp, *head;
int     gap1, gap2;
{
    struct block_rect *newblock, *b, *tmpb;
    int   i, y;

/*printf("get into block_split x = %d y = %d cover = %d first = %d first = %d\n",
            tmp->left_x, tmp->upper_y, tmp->cover, tmp->firstsubblock,
            tmp->firstnonextblock);*/
    ;

    newblock = new_block(0,0);

    b = tmp->firstsubblock;

    newblock->pairright = b->pairright;
    newblock->pairleft = b->pairleft;
    newblock->upper_y = gap1;
    newblock->lower_y = b->lower_y;
    newblock->right_x = b->left_x;
    newblock->left_x = b->right_x;
    for ( y = newblock->upper_y; y <= newblock->lower_y; y++)
        for ( i = 1; i <= b->pairleft[y][0]; i++)
            if (newblock->right_x < b->pairleft[y][i])
                newblock->right_x = b->pairleft[y][i];
    for ( y = newblock->upper_y; y <= newblock->lower_y; y++)
        for ( i = 1; i <= b->pairright[y][0]; i++)
            if (newblock->left_x > b->pairright[y][i])
                newblock->left_x = b->pairright[y][i];

    newblock->length = newblock->lower_y-newblock->upper_y+1;
    newblock->width = newblock->right_x-newblock->left_x+1;
    newblock->no_point=newblock->length/b->length;
    newblock->y = gap1;
```

【図7.9】

```
    newblock->x = newblock->right_x;
    for (i = 1; i <= b->pairright[newblock->upper_y][1] < newblock->x)
        newblock->x = b->pairright[newblock->upper_y][1];

    newblock->in_group = b->in_group;

    b->lower_y = gap1;
    b->length = b->lower_y-b->upper_y+1;
    b->on_point = b->on_point-newblock->on_points;

    add_rectblock(TEXT, newblock, head);
/* printf("newblock->previous->index = %d \n", newblock->previous->index); */

    for (b = b->next; b != NULL;) {

/*
printf("b = %d b->y1 = %d tmp->first = %d y1 = %d y2 = %d \n",
        b->index, b->upper_y, b->lower_y, tmp->firstnubblock->index,
        tmp->firstnubblock->upper_y, tmp->firstnubblock->lower_y);
*/
        if (cover_range(b->upper_y, b->lower_y, tmp->firstnubblock->upper_y,
                tmp->firstnubblock->lower_y) < 0.5){
            tmp = b->next;
            convert_newblock(newblock);
            count_newblock(TEXT);
            transfer_block(TEXT, TEXT, b, tmp, newblock);
            b = tmp;
        } else b = b->next;
    }

    if (tmp->firstnubblock->next == NULL){
        tmp->firstnubblock->index = CNP->index;
        transfer_block(TEXT, TEXT, tmp->firstnubblock, tmp, head);
        remove_textblock(tmp, head);
    } else {
        tmp->upper_y = tmp->firstnubblock->upper_y;
        tmp->lower_y = tmp->firstnubblock->lower_y;
        tmp->left_x = tmp->firstnubblock->left_x;
        for (b = tmp->firstnubblock->next; b != NULL; b = b->next){
            if (tmp->upper_y > b->upper_y)
                tmp->upper_y = b->upper_y;
            if (tmp->lower_y < b->lower_y)
                tmp->lower_y = b->lower_y;
            if (tmp->left_x > b->left_x)
                tmp->left_x = b->left_x;
            if (tmp->right_x < b->right_x)
                tmp->right_x = b->right_x;
        }
        tmp->width = tmp->right_x-tmp->left_x+1;
        tmp->length = tmp->lower_y - tmp->upper_y+1;
    }
}

/* printf("get out of block_split\n"); */
}

/*****************************************************************/
/*                                                               */
/*      split a block                                            */
/*                                                               */
/*****************************************************************/
void block_2split(head, tmp, gap1, gap2)
struct block_rect *tmp, *head;
int   gap1, gap2;
{
    struct block_rect *newblock, *b;
    int   i, y;

/*
printf("get into block_split x = %d y = %d cover = %d first = %d first = %d\n",
        tmp->left_x, tmp->upper_y, tmp->cover, tmp->firstnubblock,
        tmp->firstnubblock->index);
*/
    newblock = new_block(0,0);

    newblock->pairright = tmp->pairright;
    newblock->pairleft = tmp->pairleft;
    newblock->upper_y = gap2;
    newblock->lower_y = tmp->lower_y;

    newblock->right_x = tmp->left_x;
    newblock->left_x = tmp->right_x;
    for ( y = newblock->upper_y; y <= newblock->lower_y; y++)
        for ( i = 1; i <= tmp->pairleft[y][0]; i++)
            if (newblock->right_x < tmp->pairleft[y][i])
                newblock->right_x = tmp->pairleft[y][i];
    for ( y = newblock->upper_y; y <= newblock->lower_y; y++)
        for ( i = 1; i <= tmp->pairright[y][0]; i++)
            if (newblock->left_x > tmp->pairright[y][i])
                newblock->left_x = tmp->pairright[y][i];

    newblock->length = newblock->lower_y-newblock->upper_y+1;
    newblock->width = newblock->right_x-newblock->left_x+1;
    newblock->on_point = tmp->on_points*newblock->length/tmp->length;

    newblock->y = gap2;

    newblock->x = newblock->right_x;
    for (i = 1; i <= tmp->pairright[newblock->upper_y][0]; i++)
        if (tmp->pairright[newblock->upper_y][i] < newblock->x)
            newblock->x = tmp->pairright[newblock->upper_y][i];

    newblock->in_group = tmp->in_group;
    newblock->cover = tmp->cover;

    tmp->lower_y = gap1;
    tmp->length = tmp->lower_y-tmp->upper_y+1;
    tmp->on_point = tmp->on_point-newblock->on_points;

    add_rectblock(TEXT, newblock, head);

/*
printf("newblock->index = %d tmp->index = %d\n", newblock->index, tmp->index);
*/

/* printf("get out of block_split\n"); */
}

/*****************************************************************/
/*                                                               */
/*      split the long connected component if some gap between   */
/*                                                               */
/*****************************************************************/
int text_split(headblock, tmp, len)
struct block_rect *headblock, *tmp;
float   len;
{
    int   i, j, *local, split;
    int   upper, lower, left, right;
    int   gap1, gap2;

/* printf("get into text_split len = %f \n", len);
```

【図80】

```
printf("index = %d y=%d x=%d len = %f tmp->length = %d \n",
        tmp->index, tmp->upper_y, tmp->left_x, len, tmp->length);
*/
        split = 0;

        if ((((float)tmp->length > len*TEXT_SCALE)&&
             (((float)tmp->length <= len*(TEXT_SCALE+1))||

                upper = (tmp->upper_y-1)*reduce_factor;
                lower = tmp->lower_y*reduce_factor-1;
                left = (tmp->left_x-1)*reduce_factor;
                right = tmp->right_x*reduce_factor-1;
/*
printf("place %\n");
*/
                total = ivector(upper, lower);
                for (i=upper; i<=lower; i++){
                    total[i] = 0;
                    for (j=left; j <= right; j++)
                        if (original_p[i][j]&0x01) total[i]++;
/*      printf("[ - %d total[%] = %d \n", i, total[i]); */
                }
                for (i = upper; (total[i] <= CUPPAS(i) <= lower); i++);
                if (total[i] <= CUPP) {
                    for (gap1 = j; (total[j] <= CUPPAS(j) <= lower)&&(gap2 = j- j++);
                        if (gap2 == lower) break;
                        gap1 = (gap1/reduce_factor)+1;
                        gap2 = (gap2/reduce_factor)+1;
                        if (gap1 == gap2){
                            if (gap1 == tmp->upper_y) gap2 = tmp->upper_y+1;
                            else if (gap2 == tmp->lower_y) gap1 = tmp->lower_y-1;
                                else gap1 = gap1+1;
                        }
/*
  printf("gap 1= %d gap 2 = %d y = %d \n", gap1, gap2, tmp->left_x,
                    tmp->upper_y);
*/
                    if (tmp->firstsubblock != NULL)
                        block_split(headblock, tmp, gap1, gap2);
                    else
                        block_split(headblock, tmp, gap1, gap2);
                    split = 1;
                    break;
                }
            free_ivector(total, upper, lower);
        }
        return split;
/* printf("get out of text_split\n");      */
}


/*************************************************************************/
/********* second time text discrimination                       *********/
/*************************************************************************/

void s_text_discriminate(headblock)
struct block_rect *headblock;
{

    int   num = 0;
    long  templength = 0;
    float tmp_length;
    struct block_rect *tmp, *tmpb;

/*printf("get into s_text_discriminate \n");*/
    tmp_length = headblock->avg_height;

    if (reduce_factor > 1)
        for (tmp = headblock->firstsubblock; tmp != NULL; tmp = tmp->next)
            text_split(headblock, tmp, tmp_length);


    for (tmp = headblock->firstsubblock; tmp != NULL; ){
        if (((float)tmp->length > tmp_length*TEXT_SCALE){
            tmp->att = UNKNOWN;
            if (tmp->cover == COVER) {
                for (tmpb = tmp->firstsubblock; tmpb != NULL; tmpb = tmpb->next)
                    if (tmpb->cover == ORIGINAL) {
                        transfer_block(TEXT, NONTEXT, tmpb, tmp, tmp);
                        break;
                    }
            }
        }
        tmpb = tmp->next;
        transfer_block(TEXT, NONTEXT, tmp, headblock, headblock);
        tmp = tmpb;
    } else {
        templength += (long)tmp->length;
        num++;
        tmp->att = TEXT;
        tmp = tmp->next;
    }
}
if (num > 0)
    headblock->avg_height = (float)(templength/(float)num);
else
    headblock->avg_height = 0.0;
if (total_text < num) {    text_length = headblock->avg_height;
                           total_text = num;
                      }
/* printf("num = %d text_length = %f \n", num,  text_length); */
/*printf("get out of  s_text_discriminate \n");*/
}

/*************************************************************************/
/********* Discriminate text and nontext                         *********/
/*************************************************************************/

void text_discriminate(headblock)
struct block_rect *headblock;
{
    struct block_rect *tmp;

    i_text_discriminate(headblock);
    s_text_discriminate(headblock);
/*
    if (i_text_discriminate(headblock))
        s_text_discriminate(headblock);
    else {
        for(tmp = headblock->firstnontextblock;
                                tmp != NULL; tmp = tmp->next:
            text_discriminate(tmp);
    }
*/
}

/*************************************************************************/
/********* Count the density inside the block                    *********/
/*************************************************************************/
```

【図81】

```
void block_density(p, block)
unsigned char *p;
struct block_rect *block;
{
    int i, j, k; zero_point;

/*
printf("get into block_density \n");
*/
    sort_for_block(block);

/*
for (i = block->upper_y; i <= block->lower_y; i++){
    for (j = block->left_x; j <= block->right_x; j++){
        if (p[i][j] == ZERO) printf("0");
        if (p[i][j] == ONE) printf("1");
        if (p[i][j] == LEFT_ONE) printf("<");
        if (p[i][j] == RIGHT_ONE) printf(">");
        if (p[i][j] == LEFTRIGHT_ONE) printf("b");
    }
    printf("\n");
}
*/

    for ( zero_point = 0, i = block->upper_y; i <= block->lower_y; i++)
        for ( j = 1; j <= block->pairsize[i][0]; j++)
            for (k= block->pairright[i][j-1]+1; k < block->pairleft[i][j]; k++)
                if (p[i][k] == ONE) block->on_point++;
                else if((p[i][k]&0x02)) zero_point++;
/*printf("zero_point = %d \n", block->on_point);
*/
    block->density = ((float)(block->on_point-zero_point))/(float)(block->on_point+zero_point);
}

/*************************************************************/
/******** Horizontal solid line testing                ********/
/*************************************************************/

int if_hsolid(p, block)
unsigned char *p;
struct block_rect    *block;
{

    int    i, j;
    int    *histgram;
    float  sigma;

/*
printf("place 6\n");
*/
    histgram = (int *)ivector(block->left_x, block->right_x);

    for (sigma = 0.0, i = block->left_x; i <= block->right_x; i++){
        for ( histgram[i] = 0, j = block->upper_y; j <= block->lower_y; j++)
            if (p[j][i]&0x01) histgram[i]++;
                sigma += ((float)(histgram[i]-block->length));
                                (histgram[i]-block->length));
    }

    sigma /= block->width;
    free((int *)(histgram+block->left_x));

    block_density(p, block);
    if ((sigma <= SOLIDTHED)&&(block->density >= SOLIDDENSITY))
        return HSOLID;
    else {
        for (sigma = 0.0, i = block->left_x; i <= block->right_x; i++){
            for ( j = block->lower_y; j >= block->upper_y; j--)
                if (p[j][i]&0x02) {
                    sigma += (float)(j-block->upper_y);
                    break;
                }
        }
        sigma /= block->width;
        if (sigma <= SOLIDTHED)
            return HSOLID;
        else return UNKNOWN;
    }
}

/*************************************************************/
/******** Vertical solid line testing                  ********/
/*************************************************************/

int if_vsolid(p, block)
unsigned char *p;
struct block_rect    *block;
{
    int    i, j;
    int    *histgram;
    float sigma;

/*
printf("place 6\n");
*/
    histgram = (int *)ivector(block->upper_y, block->lower_y);
    for (sigma = 0.0, i = block->upper_y; i <= block->lower_y; i++){
        for ( histgram[i] = 0, j = block->left_x; j <= block->right_x; j++)
            if (p[i][j]&0x02) histgram[i]++;
            sigma += ((float)(histgram[i]-block->width));
                            (histgram[i]-block->width));
    }

    free((int *)(histgram+block->upper_y));

    sigma /= block->length;
    block_density(p, block);
/*printf("sigma = %f density = %f \n", sigma, block->density); */
    if ((sigma <= SOLIDTHED)&&(block->density >= SOLIDDENSITY))
        return VSOLID;
    else {
        for (sigma = 0.0, i = block->upper_y; i <= block->lower_y; i++){
            for ( j = block->right_x; j >= block->left_x; j--)
                if (p[i][j]&0x02) {
                    sigma += (((float)(block->right_x-j)*
                                (block->right_x-j)));
                    break;
                }
        }
        sigma /= block->length;
        if (sigma <= SOLIDTHED)
            return VSOLID;
        else {
            for (sigma = 0.0, i = block->upper_y;
```

【図82】

```
        for (j = block->left_x; j <=block->right_x; j++){
            if (p[i][j]!=bk01)
                sigma += ((float)((j-block->left_x)*
                          (j-block->left_x)));
        }
    sigma /= block->length;
    if (sigma <= SOLIFIED)
        return VOLID(UP);
    else
        return UNKNOWN;
}

/*.....*/
/*..... frame testing                                  .....*/
/*.....                                                .....*/

void frame_discriminate(p, block)
unsigned char **p;
struct block_rect   *block;
{
    int num, w, i, longest, j, tmplong, t;
    long sigma;

/*
printf("get into frame_dis \n");
*/
    sort_for_block(block);

    if ((block->length > 130/SCALE_RATIO)&&(block->width > 300/SCALE_RATIO)){
        t = (int)(STANDARD*((float)block->length/(float)block->width));
        w = (block->width-t) > 0 ? block->width - t : block->width;

        for (num=0, i=block->upper_y+1, sigma = 0.0;
                    i <= block->lower_y-1; i++){
            for (longest = 0, j = 1; j <= block->pairright[i][0]; j++)
                if ((tmplong=block->pairleft[i][j]-block->pairright[i][j+1])
                                                        >longest)
                    longest = tmplong;
            if (longest > w){
                sigma +=(long)(block->width-longest);
                num++;
            }
        }
        if (num > 0) sigma /= num;
        /* frame test and content test  */

        if ((num >= (int)((float)(block->length-2)*0.5))&&(sigma <= t*2))
            block->att = FRAME;
        } else if (((float)block->length < 130/SCALE_RATIO)&&
                    ((float)block->width > 300/SCALE_RATIO))
            block->att = if_hvalid(p, block);
        else if (((float)block->width < 130/SCALE_RATIO)&&
                    ((float)block->length > 300/SCALE_RATIO))
            block->att = if_vvalid(p, block);
/* printf("att = %x \n",block->att); */
}

/*.....*/
/*..... recover the covered block into original head block  .....*/
/*.....                                                     .....*/
void text_recover(head, first)
struct block_rect  *head, *first;
{
    struct block_rect *tmp, *tmpnext;
    float             tmpl;

    tmpl = (head->avg_height == 0.0) ? text_length :head->avg_height;

    for (tmp = first->firstsubblock; tmp!=NULL; ){
        if (tmp->cover == COVERED)
        {

            if (((float)tmp->length <= tmpl*TEXT_SCALE)
                                    &&(tmp->width<head->width/WIDTH_RATIO)){
                tmpnext = tmp->next;
                transfer_block(TEXT, TEXT, tmp, first, head);
                tmp = tmpnext;
            }
            else{
                if (tmp->length > tmpl*TEXT_SCALE)
                    if (text_split(first, tmp, tmpl)) continue;
                tmpnext = tmp->next;
                transfer_block(TEXT, NONTEXT, tmp, first, head);
                tmp = tmpnext;
            }
        } else tmp = tmp->next;
    }
}

/*.....*/
/*..... Recover the covered content block into original head block  .....*/
/*.....                                                             .....*/
void ntext_recover(head, first)
struct block_rect  *head, *first;
{
    struct block_rect *tmp, *tmpnext;
    float             tmpl;

    tmpl = (head->avg_height == 0.0) ? text_length :head->avg_height;

    for (tmp = first->firstnontextblock; tmp!=NULL; ){
        if (tmp->cover == COVERED)
        {
            tmpnext = tmp->next;
            if ((((float)tmp->length <= tmpl*TEXT_SCALE)
                        &&(tmp->width<head->width/WIDTH_RATIO))){
                tmpnext = tmp->next;
                transfer_block(TEXT, TEXT, tmp, first, head);
                tmp = tmpnext;
            }
            else {
                if (tmp->length > tmpl*TEXT_SCALE)
                    if(text_split(first, tmp, tmpl)) continue;
                tmpnext = tmp->next;
                transfer_block(TEXT, NONTEXT, tmp, first, head);
                tmp = tmpnext;
            }
        } else tmp = tmp->next;
    }
}

/*.....*/
/*..... Processing for one content block contained  .....*/
/*.....                                             .....*/
void one_content_contained(block, head)
```

【図83】

```
struct block_rect *block, *head;
{
    struct block_rect *tmp;

/*
printf("get into one nontextblock \n");
printf("cove = %d firstnum = %d currentnum=%d \n",
        block->cover, block->firstnumblock, block->firstnontextblock,
        block->currentnontextblock);
*/
    if (!block->cover == COVERED&&(block->firstsubblock == NULL&&
        (block->firstnontextblock->cover == ORIGINAL))
    {
        block->firstnontextblock->next = block->next;
        if (head->firstnontextblock == block){
            head->firstnontextblock = block->firstnontextblock;
            if (head->currentnontextblock == block)
                head->currentnontextblock = block->firstnontextblock;
        }
        else
            block->next->previous = block->previous;
    } else {
        if (head->currentnontextblock == block)
            head->currentnontextblock = block->firstnontextblock;
        else {
            block->next->previous = block->firstnontextblock;
            block->previous->next = block->firstnontextblock;
            block->firstnontextblock->previous = block->previous;
        }
        block->firstnontextblock->cover = 0;
        block->firstnontextblock->index = block->index;
        block->firstnontextblock->cover = 1; */
/*      free((struct block_rect *)block);
    }
/* printf("get out of  one nontextblock \n"); */
}

/*****************************************************************/
/******** Classify nontext block                         ********/
/*****************************************************************/

void nontext_classify(p, tmpb, head)
unsigned char *p;
struct block_rect   *tmpb, *head;
{
    struct block_rect  *tmpsub;

/*  printf("get into nontext_classify"); */
        if (tmpb->att != UNKNOWN) return;
        if (tmpb->firstnontextblock != NULL)
        for ( tmpsub = tmpb->firstnontextblock; tmpsub != NULL;
                                                tmpsub = tmpsub->next) {
                frame_discriminate(p, tmpsub);
                if (((tmpsub->att==(HSOLID|DOWN))||(tmpsub->att == (HSOLID|UP))||&
                    ((tmpb->cover == COVER)&&(tmpsub->cover == ORIGINAL))
                    (
                        text_recover(head, tmpb);
                        ntext_recover(head, tmpb);
                        one_nontext_contained(tmpb);  */
                    )
                else{
                    frame_discriminate(p, tmpb);
                }
/*  printf("get out nontext_classify"); */
```

```
/*****************************************************************/
/******** Classify nontext block                         ********/
/*****************************************************************/

/* printf("get into if_halftone, x = %d y = %d\n", block->left_x, block->upper_y); */
    if (block->firstsubblock == NULL) return half;
    else {
        for (num_t = 0, dot = 0, tmp = block->firstsubblock;
                                 tmp != NULL; tmp = tmp->next){
/* printf("inde %d  %d y = %d l = %d w = %d \n",
            tmp->index, tmp->left_x, tmp->upper_y,  tmp->length, tmp->width); */
            if (((float)tmp->length < HPOINT/SCALE_RATIO)&&
                ((float)tmp->width < HPOINT/SCALE_RATIO))
                dot++;
            num_t++;
        }

/* printf("dot = %d num = %d \n", dot, num_t); */
    if ((dot >= HDOT) &&(dot >= ((num_t+1)/2)))
                        /* 75 need more consideration */
        half = HPICTURE;

/*
*****************************************************************/
/******** Classify nontext block                         ********/
/*****************************************************************/

void nontext_classify(p, tmpb, head)
unsigned char *p;
struct block_rect   *tmpb, *head;
{
    struct block_rect  *tmpsub;

/*  printf("get into nontext_classify"); */
        if (tmpb->att != UNKNOWN) return;
        if (tmpb->firstnontextblock != NULL)
        for ( tmpsub = tmpb->firstnontextblock; tmpsub != NULL;
                                                tmpsub = tmpsub->next) {
                frame_discriminate(p, tmpsub);
                if (((tmpsub->att==(HSOLID|DOWN))||(tmpsub->att == (HSOLID|UP))||&
                    ((tmpb->cover == COVER)&&(tmpsub->cover == ORIGINAL))
                    (
                        text_recover(head, tmpb);
                        ntext_recover(head, tmpb);
                        one_nontext_contained(tmpb);  */
                    )
                else{
                    frame_discriminate(p, tmpb);
                }
/*  printf("get out nontext_classify"); */
```

```
/*****************************************************************/
/******** Classify nontext block                         ********/
/*****************************************************************/

/* printf("get into if_halftone, x = %d y = %d\n", block->left_x, block->upper_y); */
    if (block->firstsubblock == NULL) return half;
    else {
        for (num_t = 0, dot = 0, tmp = block->firstsubblock;
                                 tmp != NULL; tmp = tmp->next){
/* printf("inde %d  %d y = %d l = %d w = %d \n",
            tmp->index, tmp->left_x, tmp->upper_y,  tmp->length, tmp->width); */
            if (((float)tmp->length < HPOINT/SCALE_RATIO)&&
                ((float)tmp->width < HPOINT/SCALE_RATIO))
                dot++;
            num_t++;
        }

/* printf("dot = %d num = %d \n", dot, num_t); */
    if ((dot >= HDOT) &&(dot >= ((num_t+1)/2)))
                        /* 75 need more consideration */
        half = HPICTURE;

    if ((half == HPICTURE)&&(block->cover == COVER)) {
    for (tmp = block->firstsubblock; tmp != NULL; )
        if !(((float)tmp->length <= TEXT_SCALE)
                                &&(tmp->width<head->width/WIDTH_RATIO)){
            if (!tmp->cover == COVERED)&&
                                (tmp->length == HPOINT/SCALE_RATIO)){
                tmpb = tmp->next;
                transfer_block(TEXT, TEXT, tmp, block, head);
                tmp = tmpb;
            } else tmp = tmp->next;
        ntext_recover(head, block);
        one_nontext_contained(block);
    }
```

[図84]

```
printf("get out of halftone test\n");
*/
        return halft;
}

/*************************************************************************/
/******** Discriminate text inside nontext block                 ********/
/*************************************************************************/

void text_in_nontext(first)
struct block_rect *first;
{
      struct block_rect* tmp;
/* printf("get into text in nontext\n"); */
      for (tmp = first; tmp != NULL; tmp = tmp->next){
/*          if (tmp->att&HPICTURE) continue; */
          if (tmp->firstsubblock != NULL) text_discriminate(tmp);
      }
/*printf("get out of  text in nontext\n");*/

}


/*************************************************************************/
/******** Test if halftone for the nontext block under           ********/
/*************************************************************************/

void halftone_test(testblock)
struct block_rect *testblock;
{
      struct block_rect *tmp;

      for (tmp = testblock->firstnontextblock; tmp != NULL; tmp = tmp->next)
          tmp->att |= if_halftone(tmp, testblock);
}

/*************************************************************************/
/******** Search the content of the first level nontext block     ********/
/*************************************************************************/

void firstlevelnontextsearch(p, first)
unsigned char     **p;
struct block_rect    *first;
{
      struct block_rect *tmpb, *tmpbb, *tmpbbnext, *tmpl, *tmpnt;
      float  tmpl, ad;
      int    rm, y, no_total, total, i;

/*
 printf("get into firstlevelnontextsearch\n");
*/
      for (tmpb = first->firstnontextblock; tmpb != NULL; ){
/* printf("index = %d \n", tmpb->index); */
          nontext_classify(p, tmpb, first);
          if ((tmpb->firstnontextblock == NULL)&&(tmpb->firstsubblock == NULL)){
/*
 printf("in firstlevel block_in_block, x = %d y = %d \n", tmpb->left_x,
                                    tmpb->upper_y);
*/
              if ((((tmpb->length > 150/SCALE_RATIO)||(tmpb->width >= first->width/2
))
                  &&(!(tmpb->att&HSOLID))&&(!(tmpb->att&VSOLID)))||
                                                  (tmpb->att&FRAME)){
                  block_in_block(p, tmpb, first);
                  if ((tmpb->att==FRAME)||(tmpb->att&PICTURE)||(tmpb->att&TABLE))
                     {
                         text_discriminate(tmpb);
                         if (tmpb->att == FRAME){
/*                           nt_class(p, tmpb);  */
                             firstlevelnontextsearch(p, tmpb);
                         }
                     }
              }
              tmpb = tmpb->next;
          } else
              {
                if (tmpb->firstsubblock != NULL)
                   tmpb->att |= if_halftone(tmpb, first);
/*
printf("x = %d y = %d att = %x cover= %d\n", tmpb->left_x, tmpb->upper_y,
                       tmpb->att, tmpb->cover);
*/
                if ((((!(tmpb->att&PICTURE))||((tmpb->cover != COVER)&&
                                        (tmpb->firstnontextblock != NULL)))
                   for (tmpbb = tmpb->firstnontextblock;
                                tmpbb != NULL; tmpbb = tmpbb->next){
                       if ((tmpbb->firstnontextblock == NULL)
                                        &&(tmpbb->firstsubblock == NULL))
                          if ((((tmpbb->length > 150/SCALE_RATIO)||
                                        (tmpb->width >= first->width/2))&&
                              (!(tmpbb->att&HSOLID))&&(!(tmpbb->att&VSOLID)))
                              ||(tmpbb->att&FRAME)){
                               block_in_block(p, tmpbb, tmpb);
                               if (tmpbb->att&(HPICTURE|PICTURE))
                                  tmpb->att |= tmpbb->att;
                               if ((tmpbb->att == FRAME)||(tmpbb->att&PICTURE)||
                                  (tmpbb->att&TABLE))
                                  {
/*
printf("before text dis \n");
*/
/*
for (tmpl = tmpbb->firstsubblock; tmpl != NULL; tmpl = tmpl->next)
   printf("index = %d, in_group = %d\n", tmpl->index,  tmpl->in_group);
*/
                                  text_discriminate(tmpbb);
                                  text_in_nontext(tmpbb->firstnontextblock); */
                                  if (tmpbb->att&FRAME){
                                      nt_class(p, tmpbb);
                                      firstlevelnontextsearch(p, tmpbb);
                                  }
/*
printf("after text dis \n");
for (tmpl = tmpb->firstnontextblock->firstsubblock; tmpl != NULL; tmpl = tmpl->next)
   printf("index = %d, in_group = %d\n", tmpl->index,  tmpl->in_group);
*/
                                  }
                          }
                   }
                if ((tmpb->att&HPICTURE)&&(tmpb->cover == COVER)){
/*
printf("in firstlevel\n");
*/
                   sort_for_block(tmpb->firstnontextblock);

                   total = 0;
                   no_total = 0;
                   for (tmpbb = tmpb->firstsubblock; tmpbb != NULL;
                                        tmpbb = tmpbb->next){
/*
```

```
printf("index = %d l = %d \n", tmpbb->index, tmpbb->length);
*/
                        if [[[float)tmpbb->length < 150.0/SCALE_RATIO)&&
                            (tmpbb->width < tmpb->width/3.0)){
                           total += tmpbb->length;
                           no_total++;
                        }
                    }
                    if (no_total > 0)
                        tmpb->avg_height = (float)total/((float)no_total);
                    tmpl =
                        (first->avg_height == 0.0)? tmpb->avg_height
                                                  :first->avg_height;
                    ad = (tmpb->avg_height >= MPOINT/SCALE_RATIO)?
                            (MPOINT/SCALE_RATIO-2.0):MPOINT/SCALE_RATIO;
/*
 printf("tmpl = %f  \n", tmpl);
*/
                    tmpnt = tmpb->firstnontextblock;
/*
for ( y = tmpnt->upper_y; y <= tmpnt->lower_y; y++){
    printf("  y = %d  pair = %d ", y, tmpnt->pairright[y][0]);
    for (i = 1; i <= tmpnt->pairright[y][0]; i++)
        printf("%d %d ", tmpnt->pairright[y][i], tmpnt->pairleft[y][i]);
    printf("\n");
}
*/
                    for (tmpbb = tmpb->firstsubblock; tmpbb!=NULL; ){
                    for (rm = 1, y = tmpbb->upper_y; y <= tmpbb->lower_y; y++)
                        if (tmpnt->pairright[y][0] != 0){
                           if(if_overlap(tmpbb->left_x, tmpbb->right_x,
                              tmpnt->pairright[y][1],
                              tmpnt->pairleft[y][tmpnt->pairright[y][0]])){
                              rm = 0;
                              break;
                           }
                        }
/*
printf("index = %d  rm = %d \n", tmpbb->index, rm);
*/
                        if (rm == 1){
                          if ((float)tmpbb->length >= tmpl*TEXT_SCALE){
                             if (!text_split(tmpb, tmpbb, tmpl))
                                tmpbb = tmpbb->next;
                             continue;
                          }
                          if ((((float)tmpbb->length >= ad)||
                              ((float)tmpbb->width >= ad))&&
                              (tmpbb->width <= first->width/WIDTH_RATIO)){

                             tmpbbnext = tmpbb->next;
                             transfer_block(TEXT, TEXT, tmpbb, tmpb, first);
                             tmpbb = tmpbbnext;
                          } else tmpbb = tmpbb->next;
                        } else tmpbb = tmpbb->next;
                    }
                    tmpbb = tmpb->next;
                    one_nontext_contained(tmpb, first);
/*
printf("after one_nontext \n");
*/
                    tmpb = tmpbb;
                } else if ((!(tmpb->att&PICTURE))&&(tmpb->cover == COVER)){
                        text_recover(first, tmpb);
/*printf("after text cover \n");
for (tmpl = tmpb->firstnontextblock->firstsubblock; tmpl != NULL; tmpl = tmpl->next)
    printf("index = %d, in_group = %d\n", tmpl->index,  tmpl->in_group);
*/
                        ntext_recover(first, tmpb);  */
                        tmpbb = tmpb->next;
                        tmpl = tmpb->firstnontextblock;
                        one_nontext_contained(tmpb, first);
/*
printf("after one nontext \n");
for (tmpl = tmpl->firstsubblock; tmpl != NULL; tmpl = tmpl->next)
   printf("index = %d, in_group = %d\n", tmpl->index,  tmpl->in_group);
*/
                        tmpb = tmpbb;
                    }
                }
    }
/*printf("get out firstlevelnontextsearch\n");*/
}

/*****************************************************************************/
/******** Print block message                                        ********/
/*****************************************************************************/

void print_block(level, initial_block)
int        level;
struct block_rect  *initial_block;
{
       struct block_rect* tmp;
       int nextlevel, i;

      nextlevel = level+1;
      for (tmp = initial_block->firstsubblock; tmp != NULL;tmp=tmp->next){
          for ( i = 0; i <= (level-1)*8; i++) fprintf(block_file, " ");
          fprintf(block_file,"index = %d ", tmp->index);
          fprintf(block_file,
                "x=%d y=%d l=%d w=%d d=%f on=%d subblock=%d c=%d g = %d in_g = %d\n",
                  tmp->left_x, tmp->upper_y, tmp->length, tmp->width,
                  tmp->density, tmp->on_point,
                  tmp->numsubblock, tmp->cover, tmp->group, tmp->in_group);
          print_block(nextlevel, tmp);
      }
      for (tmp = initial_block->firstnontextblock; tmp != NULL;tmp=tmp->next){
          for ( i = 0; i <= (level-1)*8; i++) fprintf(block_file, " ");
          fprintf(block_file, "nontext index = %d ", tmp->index);
          fprintf(block_file,
          "x=%d y=%d l=%d w=%d d=%f on=%d  att = %x sub = %d c=%d g=%d in_g = %d\n",
                  tmp->left_x, tmp->upper_y, tmp->length, tmp->width,
                  tmp->density, tmp->on_point,  tmp->att,
                  tmp->numsubblock, tmp->cover, tmp->group, tmp->in_group);
          print_block(nextlevel, tmp);
      }
}


/*****************************************************************************/
/******** Release block                                              ********/
/*****************************************************************************/

void release_block(initial_block)
struct block_rect  *initial_block;
{
       struct block_rect *tmp1, *tmp2, *tmp11, *tmp12 ;
       int    i;

      tmp1 = initial_block->firstsubblock;
      tmp2 = initial_block->firstnontextblock;
      for (; tmp1 != NULL; ){
          tmp11 = tmp1->next;
```

[図85]

【図87】

```
/*****************************************************************************/
/*                                                                           */
/*                File: blockline.c                                          */
/*                Author: Shin-Yuan Wang                                     */
/*                Date : 1-27-91                                             */
/*                                                                           */
/*****************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
#include "blockgeneral.h"
#include "blockline.h"

void    exit();
FILE    *fopen(), *line_file;
int     vidhand;
float   real_len;
extern  float text_length;
extern  int   reduce_factor;

char  *filltext(3)= {'empty', 'overlap', 'next', 'surroundin', 'surroundout'};
int   gap_over, gap_nonover, total_gap;
int   image_row, upper_row, lower_row;
unsigned char  *original_p;
unsigned char  *text_area;

/*****************************************************************************/
/******** allocation blockpointer                                    *********/
/*****************************************************************************/

struct block_rect **new_blockpointer(ml, nh)
int ml, nh;
{
    struct block_rect **v;

    v = (struct block_rect **) malloc((nh-ml+1)*sizeof(struct block_rect*));
    if (!v) {
        printf("error: in the allocation of blockpointer\n");
        exit(1);
    }
    v = v-ml;
    return v;
}

/*****************************************************************************/
/******** sort block_rect in y order                                 *********/
/*****************************************************************************/

struct block_rect **sort_textblock(numblock, headblock)
int
struct block_rect *headblock;
{
    int    i, j, stop;
    struct block_rect *tmp;
    struct block_rect **block_y;

    *numblock = 0;

    if (headblock->attsHPICTURE){
        for (tmp = headblock->firstsubblock; tmp!= NULL; tmp = tmp->next)
            if ((((float)tmp->length >= HPOINT/SCALE_RATIO)&&
                 (((float)tmp->length <= text_len*2.0)) &&
                 ((float)tmp->width >= HPOINT/SCALE_RATIO))
                ('numblock)++;
    } else
        for (tmp = headblock->firstsubblock; tmp != NULL; tmp = tmp->next)
            if (((float)tmp->length >= HPOINT/SCALE_RATIO)&&
                (((float)tmp->length <= text_len*2.0)) &&
                ((float)tmp->width >= HPOINT/SCALE_RATIO))
                    block_y[i++] = tmp;
    if (*numblock == 0) return NULL;
    block_y = new_blockpointer(0, (*numblock)-1);

    if (headblock->attsHPICTURE){
        for (tmp = headblock->firstsubblock, i = 0; tmp != NULL; tmp = tmp->next)
            if ((((float)tmp->length >= HPOINT/SCALE_RATIO)&&
                 (((float)tmp->length <= text_len*2.0)) &&
                 ((float)tmp->width >= HPOINT/SCALE_RATIO)){
                    block_y[i++] = tmp;
            }
    } else
        for (tmp = headblock->firstsubblock, i = 0; tmp != NULL; tmp = tmp->next){
            block_y[i++] = tmp;
        }

    for ( i = 0; i < *numblock ; i++){
        for (j = 0, stop = 0; j < *numblock -1; j++)
            if (block_y[j]->upper_y > block_y[j+1]->upper_y){
                tmp = block_y[j+1];
                block_y[j+1] = block_y[j];
                block_y[j] = tmp;
                stop = 1;
            }
        if (stop == 0) break;
    }

    return block_y;
}

/*****************************************************************************/
/******** plot area                                                  *********/
/*****************************************************************************/

unsigned char **plot_area(num, s_block, nrl, nrh, ncl, nch)
int num;
struct block_rect **s_block;
int nrl, nrh, ncl, nch;
{
    int i, k, m;

    text_area = cmatrix(nrl, nrh, ncl, nch);

    for ( i = 0; i < num; i++)
        for (k = s_block[i]->upper_y; k <= s_block[i]->lower_y; k++)
            for (m = s_block[i]->left_x; m <= s_block[i]->right_x; m++)
                text_area[k][m] = 1;
}

/*****************************************************************************/
/******** test if one block embrace the other, or vice versa         *********/
/*****************************************************************************/

int min(first, second)
struct block_rect *first, *second;
{
    if ((second->left_x <= first->left_x)&&(second->right_x >= first->right_x))
        return 1;
```

【図88】

```
    if (((first->left_x <= second->left_x)&&(first->right_x >= second->right_x))
        return 1;

    return 0;
}

/*.....test if y range overlap                                    .....*/
/*.....                                                           .....*/

int ycouch(first, second)
struct block_rect *first;
struct linelock *second;
{
    int i;

    if ((cover_range(first->upper_y, first->lower_y,
        second->lower_y, second->lower_y)>LINEOVERLAP)||
        (cover_range(second->upper_y, second->lower_y,
            first->upper_y, first->lower_y)>LINEOVERLAP))
        return 1;
    else
        return 0;
}

/*.....                                                           .....*/
/*.....line filling test                                          .....*/
/*.....                                                           .....*/

fillcondition fill_test(x, current_block, head, widthfill)
int x;
struct block_rect *current_block, *head;
struct block_rect *widthfill;
{
    int         i, j, intest;
    int         begin, end;
    fillcondition   fill = EMPTY;
    struct block_rect *previous_block;

    begin = current_block->left_x;
    end = current_block->right_x;

    previous_block = NULL;
    for ( ; begin <= end; previous_block = widthfill[i], i++){
        if ( (widthfill[i] == NULL)||(widthfill[i] == previous_block)) continue;
        if ( (widthfill[i]->lower_y > current_block->upper_y) ||
             (((float)(current_block->lower_y-widthfill[i]->upper_y)
                                        <= text_len*TRUE /*........ Chain the nontext line block
            (current_block->width >= widthfill[i]->width - DOT_WIDTH))){
            intest = min(current_block, widthfill[i]);
            if (intest == 1){
                fill = SURROUNDIN;
                break;
            } else if (intest == 2) {
                fill = SURROUNDOUT;
                converto_headblock(current_block);
                transfer_block(next, TEXT,widthfill[i], head,current_block);
                for (j = i+1; widthfill[j] != widthfill[j-1]; j++)
                    widthfill[j] = NULL;
                widthfill[i] = NULL;
                i = j-1;
                continue;
            } else {
                fill = OVERLAP;
                continue;
            }
        } else {
            fill = NEXT;
            break;
        }
    }
    *x = i;
    return fill;
}

/*.....                                                           .....*/
/*..... Allocate new line block                                   .....*/
/*.....                                                           .....*/

struct linelock *new_inblock(nl, nh)
int nl, nh;
{
    int i;
    struct linelock *v;

    v = (struct linelock *) malloc((unsigned)(nh-nl+1)*
                                    sizeof(struct linelock));

    if (!v){
        printf("allocation error: line block \n");
        exit(1);
    }
    v -= nl;
    for ( i = nl; i <=nh; i++) {
        v[i].first_block = v[i].first_block = NULL;
        v[i].current_line = v[i].currentnontextline = NULL;
        v[i].firstnontextline = v[i].currentnontextline = NULL;
        v[i].previous = v[i].next = NULL;
        v[i].upper_y = v[i].lower_y = v[i].left_x = v[i].right_x = 0;
        v[i].width = v[i].length = 0;
        v[i].group = 0;
        v[i].att = UNKNOWN;
        v[i].nameblock = 0;
        v[i].index = 0;
        v[i].side_gap = 0;
        v[i].under_caption = NULL;
        v[i].r_caption = v[i].l_caption = 0;  */
    }
    return v;
}

struct linelock *new_nontextline(headline)
struct linelock *headline;
{
    struct linelock *n;

    n = new_inblock(0,0);
    if (headline->firstnontextline == NULL)
        headline->firstnontextline = n;
    else{
        headline->currentnontextline->next = n;
        n->previous = headline->currentnontextline;
    }
```

【図89】

```
        headline->currentnontextline = n;
        return n;
}

/***************************************************************************/
/*******   Chain the line block                                    ********/
/***************************************************************************/

struct lineblock *new_line(headline)
struct lineblock *headline;
{
        struct lineblock *n;

        n = new_lnblock(0,0);
        if (headline->firstsubline == NULL)
            headline->firstsubline = n;
        else{
            headline->currentsubline->next = n;
            n->previous = headline->currentsubline;
            }
        headline->currentsubline = n;
        return n;
}

/***************************************************************************/
/******* Add block into line block forward                          ********/
/***************************************************************************/

void fadd_block_into_line(rect_block, head_block, line_block)
struct block_rect *rect_block, *head_block;
struct lineblock  *line_block;
{
        struct block_rect *tmpblock;

        remove_textblock(rect_block, head_block);
        head_block->numsubblock--;
        if (line_block->first_block == NULL){
            line_block->first_block = line_block->current_block = rect_block;
            rect_block->previous = rect_block->next = NULL;
            line_block->left_x = rect_block->left_x;
            line_block->right_x = rect_block->right_x;
            line_block->upper_y = rect_block->upper_y;
            line_block->lower_y = rect_block->lower_y;
            line_block->numsubblock++;
        } else {
                line_block->current_block->next = rect_block;
                rect_block->previous = line_block->current_block;
                line_block->current_block = rect_block;
                rect_block->next = NULL;
                if (rect_block->left_x < line_block->left_x)
                        line_block->left_x = rect_block->left_x;
                if (rect_block->right_x > line_block->right_x)
                        line_block->right_x = rect_block->right_x;
                if (rect_block->upper_y < line_block->upper_y)
                        line_block->upper_y = rect_block->upper_y;
                if (rect_block->lower_y > line_block->lower_y)
                        line_block->lower_y = rect_block->lower_y;
                line_block->numsubblock++;
              }
}

/***************************************************************************/
/******* Add block into line block backward                         ********/
/***************************************************************************/

void badd_block_into_line(rect_block, head_block, line_block)
struct block_rect *rect_block, *head_block;
struct lineblock  *line_block;
{
        struct block_rect *tmpblock;

        remove_textblock(rect_block, head_block);
        head_block->numsubblock--;
        if (line_block->first_block == NULL){
            line_block->first_block = line_block->current_block = rect_block;
            rect_block->previous = rect_block->next = NULL;
            line_block->left_x = rect_block->left_x;
            line_block->right_x = rect_block->right_x;
            line_block->upper_y = rect_block->upper_y;
            line_block->lower_y = rect_block->lower_y;
            line_block->numsubblock++;
        } else {
                line_block->first_block->previous = rect_block;
                rect_block->next = line_block->first_block;
                line_block->first_block = rect_block;
                rect_block->previous = NULL;
                if (rect_block->left_x < line_block->left_x)
                        line_block->left_x = rect_block->left_x;
                if (rect_block->right_x > line_block->right_x)
                        line_block->right_x = rect_block->right_x;
                if (rect_block->upper_y < line_block->upper_y)
                        line_block->upper_y = rect_block->upper_y;
                if (rect_block->lower_y > line_block->lower_y)
                        line_block->lower_y = rect_block->lower_y;
                line_block->numsubblock++;
              }
}

/***************************************************************************/
/******** test if a gap between two rectblock                      ********/
/***************************************************************************/

int if_gap(block1, block2, headblock)
struct block_rect *block1, *block2, *headblock;
{
        int center_x, i, count1, count2;
        int bound1, bound2;
        int len, maxcount, x_maxcount;


        if (block2->left_x <= block1->right_x+2) return 0;

        bound1 = min(block1->upper_y, block2->upper_y);
        bound2 = max(block1->lower_y, block2->lower_y);

        for (center_x = block1->right_x+1; center_x < block2->left_x; center_x++){
            for ( i = bound1, count1 = 0; i <= bound2; i++)
                if (original_p[i][center_x]&GAPMARK) {
                   printf("%x ", original_p[i][center_x]);
                   count1++;
                }

            if (count1 == (bound2-bound1+1)) return 1;
        }

        len = (MAXFONT/SCALE_RATIO < 15*text_len)? MAXFONT/SCALE_RATIO:15*text_len;
        maxcount = 0;
        x_maxcount = 0;
```

【図90】

```
for (center_x = block2->right_x+1; center_x < block2->left_x; center_x++){
    for (count = 0, i = (bound1+bound2)/2;
            i <= headblock->lower_y; i++){
        if (original_p[i][(center_x-1)&0x01]) break;
        else count++;
    }
    if (count > maxcount) {
        maxcount = count;
        x_maxcount = center_x;
    }
}
if (maxcount >= len){
printf("len = %d max_count = %d x = %d y = %d \n", len, maxcount, x_maxcount,
                (bound1+bound2)/2 );
    for (i = (bound1+bound2)/2, count1 = 1; count1 <= maxcount; count1++, i+=2
        original_p[i][x_maxcount] |= CAPMARK;
    return 1;
}
else
    return 0;
*/
}

/*****************************************************************/
/******** Group block into line                          ********/
/*****************************************************************/

void group_connected_block(stopx, column, currentlineblock, headblock, widthfill)
int stopx, column;
struct linablock *currentlineblock;
struct block_rect *widthfill;
struct block_rect *headblock;
{
    int   x, count, i;
    int   first, second;
    struct block_rect *tmp_block;

    /* add the first block forward */
    add_block_into_line(widthfill[stopx], headblock, currentlineblock);
    for (x = stopx -1, tmp_block = widthfill[stopx]; x >= 0; x--){
        if (widthfill[x] == tmp_block) {
            widthfill[x] = NULL;
            continue;
        }

        count = 0;
        for (; (x >= 0)&&(count<widthsel); x--, count++){
            if (widthfill[x] == NULL) continue;
            if (!(ycouch(widthfill[x], currentlineblock))) {
                for ( ; (x > 0)&&(widthfill[x-1] == widthfill[x]);
                    x--, count++);
            } else break;
        }
        if ((count < widthsel)&&(count > 0)){
            total_gap += count;
            gap_removed++;
        } else
            gap_over++;

        if (count >= widthsel)
            currentlineblock->side_gap |= LEFTMAP;
        else
            if (x < 0)||(count >= widthsel)) break;
/* printf("index = %d \n", currentlineblock->index);*/

        if (!(!if_gap(widthfill[x], currentlineblock->first_block, headblock))){
            &&(currentlineblock->right_x-currentlineblock >= text_len*10)) {

            currentlineblock->side_gap |= LEFTMAP;
            break;
        }

        add_block_into_line(widthfill[x], headblock, currentlineblock);
        tmp_block = widthfill[x];
        widthfill[x] = NULL;
    }

    for (x = stopx+1; tmp_block = widthfill[stopx]; x < column; x++){
        if (widthfill[x] == tmp_block) {
            widthfill[x] = NULL;
            continue;
        }

        count = 0;
        for (; (x < column)&&(count<widthsel); x++, count++){
            if (widthfill[x] == NULL) continue;
            if (!(ycouch(widthfill[x], currentlineblock))) {
                for ( ; (x < column-2)&&(widthfill[x] == widthfill[x+1])
                        &&(count < widthsel); x++, count++);
            } else break;
        }

        if ((count < widthsel)&&(count >0)){
            total_gap += count;
            gap_removed++;
        } else
            gap_over++;

        if (count >= widthsel)
            currentlineblock->side_gap |= RIGHTMAP;

        if ((x >= column))||(count >= widthsel))
            break;
    }
/* printf("index = %d image_row = %d ", currentlineblock->index, image_row); */

    if (!(!if_gap(currentlineblock->current_block, widthfill[x], headblock))&
        &&(currentlineblock->right_x-currentlineblock->left_x > text_len*10))

        currentlineblock->side_gap |= RIGHTMAP;
        break;
    }

    add_block_into_line(widthfill[x], headblock, currentlineblock);
    tmp_block = widthfill[x];
    widthfill[x] = NULL;
    continue;
}
```

【図91】

```
    currentlinkblock->att = TEXT;
    currentlinkblock->width = currentlinkblock->right_x -
        currentlinkblock->left_x + 1;
    currentlinkblock->length = currentlinkblock->lower_y -
        currentlinkblock->upper_y + 1;

    width[11(stage] = NULL;    currentlinkblock->upper_y = 1;
}

/*.....*/
/*..... test if one line totally inside the other line .....*/
/*.....*/

int line_inside(first, second)
struct linkblock *first, *second;
{
  if (((first->left_x >= second->left_x)&&(first->right_x<=
      (first->upper_y >= second->upper_y)&&(first->lower_y <= second->lower_y))
    return 1;
  else
    return 0;
}

/*.....*/
/*..... check if the line below the picture, bitcure or table .....*/
/*.....*/

struct block_rect *if_caption(line, block)
struct linkblock *line;
struct block_rect *block;
{
  struct block_rect *tmp;

  for ( tmp = block->firstnoatextblock; tmp != NULL; tmp = tmp->next){
    if (tmp->upper_y >= line->upper_y) break;
    if ((tmp->attribute!=PICTURE)&&(tmp->attribute!=TABLE)) continue;
    else if ((cover_range(line->left_x, line->right_x, tmp->left_x,
              tmp->right_x) >= 1.0) &&(line->upper_y <= tmp->lower_y+text_len)&&
              (line->upper_y>tmp->lower_y)) {d_y1 = %d y2 = %d',
printf('.....is caption block-index =%d x1 = %d x2 = %d y1 = %d',
  tmp->index, tmp->left_x, tmp->right_x, tmp->upper_y);
printf('line-index= %d x1 = %d x2 = %d y1 = %d\n',
  line->index, line->left_x, line->upper_y, line->right_x, line->lower_y);
        line->att = CAPTION;
        line->under_caption = tmp;
        line->l_caption = tmp->left_x;
        line->r_caption = tmp->right_x;
        return tmp;
      }
  }
  return NULL;
}

/*.....*/
/*..... test if graphs between the linkblock .....*/
/*.....*/

inc if_graphs(first, second)
struct linkblock *first, *second;
{
    int center_x, count, i;
    int bound1, bound2;

    bound1 = min(first->upper_y, second->upper_y);
    bound2 = max(first->lower_y, second->lower_y);

    for (center_x = (first->right_x+1) center_x = second->left_x, center_x->( /*
        for ( i = bound1, count = 0; i<=bound2; i++)
            if (original_p[i][center_x]==BLACK) count++;
```

```
        if (count == (bound2-bound1+1)) return 1;
        return 0;
    }
}

/*.....*/
/*..... test if vertical position in the same level .....*/
/*.....*/

int same_line(firstline, secondline, head)
struct linkblock *firstline, *secondline;
struct block_rect *head;
{
    struct block_rect *tmp1, *tmp2;
    struct linkblock *tmpline;
    int   if_neighbor;
    int   x1, y1, x2, y2, i, j;

    tmp1 = if_caption(firstline, head);
    tmp2 = if_caption(secondline, head);
/* printf("after if_captionis"); */
/* printf("after if_captionis"); */
    if (((tmp1 != NULL)&&(tmp2 == NULL))||((tmp1 == NULL)&&(tmp2 != NULL))||
        ((tmp1 != NULL)&&(tmp2 != NULL)&&(tmp1 != tmp2)))
        return 0;

    if ((firstline->left_x > secondline->left_x){
      if ((if_graphs(secondline, firstline)) return 0;
      x1 = secondline->right_x;
      y1 = secondline->upper_y;
      x2 = firstline->left_x-1;
      y2 = firstline->lower_y;
    } else {
        if ((if_graphs(firstline, secondline)) return 0;
        x1 = firstline->right_x+1;
        y1 = min(firstline->upper_y, secondline->upper_y);
        x2 = secondline->left_x;
        y2 = max(firstline->lower_y, secondline->lower_y);
    }
    for ( i = x1; i <= x2; i++)
      for (j = y1; j <= y2; j++)
        if (original_p[j][i]==BLACK) return 0;
    for (if_neighbor = 0, tmpline = secondline->previous; tmpline != NULL;
                                    tmpline = tmpline->previous){
      if (tmpline->upper_y < min(firstline->upper_y, secondline->upper_y)
                              -(int)(4.0*text_len))
        break;
      else if ((if_overlap(tmpline->left_x, tmpline->right_x,
            firstline->left_x, firstline->right_x)&&
          (if_overlap(tmpline->left_x, tmpline->right_x, secondline->left_x,
              secondline->right_x))&&
          (tmpline->lower_y > min(firstline->upper_y, secondline->upper_y)
                              -(int)(2.0*text_len)))){
        if(neighbor < 1)
          neighbor = 1;
        break;
      }
    }
    if (if_neighbor == 0){
    for (tmpline = secondline->next; tmpline != NULL;
                                    tmpline = tmpline->next){
      if ((if_overlap(tmpline->left_x, tmpline->right_x, x1, x2))&&
          (if_overlap(tmpline->upper_y, tmpline->lower_y, y1, y2)))
```

【図92】

```
    return 0;
}

/*
    if (tapline->upper_y > max(firstline->lower_y, secondline->lower_y)
            +(int)(2.0*text_len))
        break;
    else if ((!_overlap(tapline->left_x, tapline->right_x,
            firstline->left_x, firstline->right_x)&&
        !_overlap(tapline->left_x, tapline->right_x, secondline->left_x,
            secondline->right_x))){
        if_neighbor = 1;
        break;
    }
}
if (!if_neighbor == 1){
printf("%3d:%3d y-hi. %d bl-%d y-hi %d ri-%d y1=%d y2=%d l = %f\n",
    firstline->index, firstline->upper_y, firstline->lower_y,
    secondline->index, secondline->upper_y, secondline->lower_y,
    tapline->index, tapline->left_x,
    tapline->upper_y, tapline->right_x, text_len);
printf("neighbor = %d \n", if_neighbor);
}
    if (!if_neighbor == 1){
    if ((cover_range(firstline->upper_y, firstline->lower_y,
            secondline->upper_y, secondline->lower_y) >= LINEOVERLAP)||
        (cover_range(secondline->upper_y, secondline->lower_y,
            firstline->upper_y, firstline->lower_y ) >= LINEOVERLAP)){
        (min(firstline->width, secondline->width) = text_len*2);  */
printf("firstline = %d secondline = %d tapline = %d \n", firstline->index,
    secondline->index, tapline->index);
        return 1;
    }
}

    return 0;
}

/*************************************************************************/
/******** sort line                                                ********/
/*************************************************************************/
void sort_line(firstlineblock)
struct linelock *firstlineblock;
{
    struct linelock *tmp;
    struct linelock *line_x;
    int    num, i, j, stop;

    if (firstlineblock->firstsubline == NULL) return;
    for (num = 0, tmp = firstlineblock->firstsubline; tmp != NULL; tmp = tmp->next)
        num++;

    if (num > 0) {
        line_x = new_linepointer(0, num-1);
        for (i = 0, tmp = firstlineblock->firstsubline; tmp != NULL; tmp = tmp->next)
            line_x[i++] = tmp;
    }

    /* extra testing */

    sort_linepointer(line_x, num);

    firstlineblock->firstsubline = line_x[0];
    firstlineblock->currentsubline = line_x[num-1];
    if (num > 1){
        line_x[0]->next = line_x[1];
        line_x[0]->previous = NULL;
        line_x[num-1]->previous = line_x[num-2];
        line_x[num-1]->next = NULL;
        line_x[num-1]->index = num;
        line_x[0]->index = 1;
    }
    for ( i = 1; i < num-1; i++){
        line_x[i]->previous = line_x[i-1];
        line_x[i]->next = line_x[i+1];
        line_x[i]->index = i+1;
    }
    free((struct linelock **)line_x);
}

/*************************************************************************/
/******** combine fully overlapped block                           ********/
/*************************************************************************/
void combine_full_overlap_line(first, firstblock)
struct linelock *first;
struct block_rect *firstblock;
{
    struct linelock *tapline, *subtapline, *tmp;
    int i;
    float gapmean;

    if (first->firstsubline == NULL) return;
    if (gap_nomover != 0) {
        gapmean = (float)total_gap/(float)gap_nomover;
printf("widthstd = %d total_gap = %d gap_nomover = %d gap_over=%d mean = %f\n",
        widthstd, total_gap, gap_nomover, gap_over, gapmean);

        widthstd += (int)((float)gap_over*widthstd/
                ((float)(widthstd+gapmean)*(float)(gap_nomover)));
    }
printf("widthstd = %d total_gap = %d gap_nomover = %d gap_over=%d imag_rowwidth",
    widthstd, total_gap, gap_nomover, gap_over, image_row);

    sort_line(first);
    for (tapline=first->firstsubline; tapline != NULL; tapline = tapline->next)
        for (subtapline=tapline->firstsubline; subtapline != NULL; ){
            if (!if_overlap(subtapline->upper_y, subtapline->lower_y,
                    tapline->upper_y, tapline->lower_y)){
                if (!if_overlap(subtapline->left_x, subtapline->right_x,
                        tapline->left_x, tapline->right_x)) {

                    if (line_inside(tapline, subtapline)){
                        line_inside(subtapline, tapline);
                        tmp = subtapline->previous;
                        line_transfer(subtapline, first, tapline, TEXT);
                        subtapline = tmp;
                        continue;
                    }
                }
                if (case_line(subtapline, tapline, firstblock)){
                tmp = subtapline->previous;
                line_transfer(subtapline, first, tapline, TEXT);
                subtapline = tmp;
                continue;
                }
            }
            else {
```

【図93】

```
if ((subtmpline->right_x < tmpline->left_x)&&
    (subtmpline->right_x >= tmpline->left_x-widthat3))&&
    (((tmpline->side_gap&LEFTGAP))||
     ((subtmpline->side_gap&RIGHTGAP))))||
    ((tmpline->right_x < subtmpline->left_x)&&
     (tmpline->right_x >= subtmpline->left_x-widthat4))&&
     (((subtmpline->side_gap&LEFTGAP))||
      ((subtmpline->side_gap&RIGHTGAP))))
print("index 1 = %d index 2 = %d\n", subtmpline->index, tmpline->index);
                if (same_line(subtmpline, tmpline, firstblock)){
print("*** index 1 = %d index 2 = %d\n", subtmpline->index, tmpline->index);
                    tmp = subtmpline->previous;
                    line_transfer(subtmpline, first, tmpline, TEXT);
                    subtmpline = tmp;
                    continue;
                }
            }
        }
*/
        subtmpline = subtmpline->previous;
    }
for (tmpline=first->firstmpline, i = 1; tmpline != NULL;
        tmpline = tmpline->next, i++)
    tmpline->index = i;
}

/*****************************************************************/
/*********  Formation of line block.                     *********/
/*****************************************************************/

void line_block(headlineblock, firstblock, column)
struct linelock *headlineblock;
struct block_rect *firstblock;
int   column;
{
    int   i, j, stopx, sort_index, num;
    int   numlineblock;
    struct block_rect *widthfill;
    struct block_rect *current_block;
    struct block_rect *sorted_block;
    struct linelock   *templineblock;
    struct linelock   fill;

    total_gap = 0; gap_nonover = 0;
    upper_row = firstblock->upper_y;
    lower_row = firstblock->lower_y;
    upper_row = (upper_row-1)*reduce_factor;
    lower_row = lower_row*reduce_factor-1;
    if (lower_row >= image_row) lower_row = image_row;

    numlineblock = 0;
    text_len = (firstblock->avg_height != 0.0)?
                        firstblock->avg_height:text_length;

    widthard = (int)((float)text_len*WIDTHRATIO+0.5);

print("len = %f std = %d \n", text_len, widthard);
    widthfill = new_blockpointer(0, column-1);
    for ( i = 0; i < column; widthfill[i++] = NULL);
    sorted_block = sort_textblock(&num, firstblock);
    text_area = plot_area(num, sorted_block, firstblock->upper_y,
                          firstblock->lower_y, firstblock->left_x,
                          firstblock->right_x);

for (sort_index = 0; sort_index < num; ){
    fill=fill_textblock(stopx, sorted_block[sort_index], firstblock, widthfill);

    /* group = new line */

    if (fill == NEXT){
      templineblock = new_line(headlineblock);
      templineblock->index = ++numlineblock;
      group_connected_block(stopx, column, templineblock, firstblock,
                                                    widthfill);
      continue;
    } else if (fill == SURROUNDIN) {
          converte_headblock(widthfill[stopx]);
          transfer_block(TEXT, TEXT, sorted_block[sort_index],
                         firstblock, widthfill[stopx]);
          } else {
              for ( j = sorted_block[sort_index]->left_x;
                    j <= sorted_block[sort_index]->right_x; j++)
                widthfill[j] = sorted_block[sort_index];
          }
      sort_index++;
    }

for ( i = 0; i < column; i++){
  if (widthfill[i] != NULL){
    templineblock = new_line(headlineblock);
    stopx = i;
    templineblock->index = ++numlineblock;
    group_connected_block(stopx, column, templineblock,firstblock,
                                                    widthfill);
    i = templineblock->right_x;
  }
}
combine_full_overlap_line(headlineblock, firstblock);
printf("********** after first combine **********\n");
combine_full_overlap_line(headlineblock, firstblock);

free_cmatrix(text_area, firstblock->upper_y, firstblock->lower_y,
             firstblock->left_x, firstblock->right_x);

free((struct block_rect**)widthfill);
free((struct block_rect**)sorted_block);
}

/*****************************************************************/
/*********  group line for the block contain more than one group area  *********/
/*****************************************************************/

void line_group(firstlineblock, firstblock)
struct linelock *firstlineblock;
struct block_rect *firstblock;
{
    int i, x, y, j, center_y;
    struct linelock **tmpline;
    struct block_rect *tmpblock, *nextblock;
    struct wklinelock *tmpwkls;

    tmpline = new_linepointer(1, firstblock->group);
    for ( i = 1; i <= firstblock->group; i++)
      tmpline[i] = new_line(firstlineblock);

    for (tmpblock = firstblock->firstsubblock; tmpblock != NULL; ){
      nextblock = tmpblock->next;
      add_block_into_line(tmpblock, firstblock, tmpline[tmpblock->in_group]);
      tmpblock = nextblock;
    }
```

[図94]

```
for (tmpwhite = firstblock->firstwhite; tmpwhite != NULL; tmpwhite = tmpwhite->nextd)
printf("white index = %d x1 = %d y1 = %d x2 = %d y2 = %d \n",
                tmpwhite->index, tmpwhite->left_x, tmpwhite->upper_y,
                tmpwhite->right_x, tmpwhite->lower_y);

    for (tmpwhite = firstblock->firstwhite, i = 1; tmpwhite != NULL;
                                        i++, tmpwhite = tmpwhite->next)
        {

         if (tmpline[i]->first_block == NULL){
              tmpline[i]->left_x = (tmpwhite->right_x+tmpwhite->left_x)/2;
              tmpline[i]->right_x = (tmpwhite->right_x+tmpwhite->left_x)/2;
              tmpline[i]->upper_y = tmpwhite->upper_y+1;
              tmpline[i]->lower_y = tmpwhite->lower_y-1;
         }

         if (((tmpline[i]->lower_y-tmpline[i]->upper_y+1) < HPOINT/SCALE_RATIO)&&
                             (tmpline[i]->first_block != NULL)){
            for (y = tmpline[i]->upper_y-1, j = 1; (j <= STANDARDFONT/2)&&(y > 0);
                                                        y--, j++){
                for (x = tmpline[i]->right_x; x <= tmpline[i]->left_x; x++)
                    if (original_p[y][x]&0x01)
                        break;
                if (original_p[y][x]&0x01)  break;
            }
            if (tmpline[i]->upper_y >= y+3) tmpline[i]->upper_y = y+2;
            for (y = tmpline[i]->lower_y+1, j = 1; (j <= STANDARDFONT/2)&&(y > 0);
                                                        y++, j++){
                for (x = tmpline[i]->right_x; x <= tmpline[i]->left_x; x++)
                    if (original_p[y][x]&0x01) break;
                if (original_p[y][x]&0x01)  break;
            }
            if (tmpline[i]->upper_y+3 <= y) tmpline[i]->lower_y = y-2;
         }


         for (x = tmpline[i]->left_x; x > tmpwhite->left_x+1; x--)
             for (y = tmpline[i]->upper_y+2; y <= tmpline[i]->lower_y-2; y++)
                 if (original_p[y][x]&0x01){
                                    tmpline[i]->left_x = x;
                                    break;
                                }

         for (x = tmpline[i]->right_x; x < tmpwhite->right_x-1; x++)
             for (y = tmpline[i]->upper_y+2; y <= tmpline[i]->lower_y-2; y++)
                 if (original_p[y][x]&0x01){
                                    tmpline[i]->right_x = x;
                                    break;
                                }
         tmpline[i]->length = tmpline[i]->lower_y-tmpline[i]->upper_y+1;
         tmpline[i]->width = tmpline[i]->right_x-tmpline[i]->left_x+1;

        }
    firstlineblock->firstsubline = tmpline[1];
    firstlineblock->currentsubline = tmpline[firstblock->group];
    for ( i = 1; i <= firstblock->group; i++){
        tmpline[i]->att = TEXT;
        tmpline[i]->index =  i;
    }

    tmpline[1]->next = tmpline[2];
    tmpline[firstblock->group]->previous = tmpline[firstblock->group-1];
    for ( i = 2; i < firstblock->group; i++) {
        tmpline[i]->next = tmpline[i+1];
        tmpline[i]->previous = tmpline[i-1];
    }
    free((struct lineblock **)(tmpline+1));


/*************************************************************************/
/******** group line for the picture block containing more than one group area**/
/*************************************************************************/

void line_picture(firstlineblock, firstblock)
struct lineblock *firstlineblock;
struct block_rect *firstblock;
{
    int i, j,  line;
    struct lineblock **tmpline;
    struct block_rect *tmpblock, *nextblock;

    tmpline = new_linepointer(1, firstblock->group);
    for ( i = 1; i <= firstblock->group; i++)
        tmpline[i] = new_lnblock(0,0);

    for (tmpblock = firstblock->firstsubblock; tmpblock != NULL;  ){
        nextblock = tmpblock->next;
        fadd_block_into_line(tmpblock, firstblock, tmpline[tmpblock->in_group]);
        tmpblock = nextblock;
     }


    for ( i = 1, line = 0; i <= firstblock->group; i++){
        if (tmpline[i]->first_block != NULL){
            firstlineblock->firstsubline = tmpline[i];
            firstlineblock->currentsubline = tmpline[i];
            tmpline[i]->length = tmpline[i]->lower_y-tmpline[i]->upper_y+1;
            tmpline[i]->width = tmpline[i]->right_x-tmpline[i]->left_x+1;
            tmpline[i]->att = TEXT;
            tmpline[i]->index = ++line;
            for ( j = i+1; j <= firstblock->group; j++){
                if (tmpline[j]->first_block != NULL){
                    tmpline[j]->previous = firstlineblock->currentsubline;
                    firstlineblock->currentsubline = tmpline[j];
                    tmpline[j]->next = NULL;
                    tmpline[j]->length = tmpline[j]->lower_y-tmpline[j]->upper_y+1;
                    tmpline[j]->width = tmpline[j]->right_x-tmpline[j]->left_x+1;
                    tmpline[j]->att = TEXT;
                    tmpline[j]->index = ++line;
                } else free((struct lineblock *)tmpline[j]);
            }
            break;
         } else free((struct lineblock *)tmpline[i]);
    }
    free((struct lineblock **)(tmpline+1));
}

/*************************************************************************/
/******** make gap on the left and right side of picture        ***********/
/*************************************************************************/

void make_gap(firstblock)
struct block_rect *firstblock;
{
    struct block_rect *tmp;
    int    x, y, count;


    for (tmp = firstblock->firstnontextblock; tmp != NULL; tmp = tmp->next){
        if !((tmp->att&(HPICTURE|FRAME|PICTURE|TABLE|VEQLID))){
```

【図95】

```
printf("make_gap on %d x = %d y = %d \n", tmp->index, tmp->left_x, tmp->upper_y);
        x = tmp->left_x -1;
        x = tmp->left_x-1;
/*      for ( x = tmp->left_x-1; x >= max((tmp->left_x-(int)text_length),
                        (firstblock->left_x)); x--){
*/
        for (y = tmp->upper_y; y <= tmp->lower_y; y++)
            if (original_p[y][x&0x07]) count++;
printf("count = %d x = %d y1 = %d y2 = %d y:end \n", count, x, tmp->upper_y, tmp->low
er_y);
            if (count <= NOISE_NO){
                for (y = tmp->upper_y; y <= tmp->lower_y; y++)
                    original_p[y][x] |= GAPMARK;
                for (y = tmp->lower_y+1; y <= firstblock->lower_y ; y++)
                    if (original_p[y][x&0x07]) break;
                    else original_p[y][x] |= GAPMARK;
        }
/*      for (x = tmp->right_x +1; x <= min((tmp->right_x+(int)text_length),
                        (firstblock->right_x)); x++){
*/
        count = 0;
        for (y = tmp->upper_y; y <= tmp->lower_y; y++)
            if (original_p[y][x&0x07]) count++;
printf("count = %d x = %d y1 = %d y2 = %d y:end \n", count, x, tmp->upper_y, tmp->low
er_y);
            if (count <= NOISE_NO){
                for (y = tmp->upper_y; y <= tmp->lower_y; y++)
                    original_p[y][x] |= GAPMARK;
                for (y = tmp->lower_y+1; y <= firstblock->lower_y ; y++)
                    if (original_p[y][x&0x07]) break;
                    else original_p[y][x] |= GAPMARK;
            }
        }
/*  }
*/
}

/*.....
/*..... Initialize the formation of line block
/*.....

void ini_line_block(firstlineblock, firstblock, column, orig_row, o_p)
struct linablock *firstlineblock;
struct block_rect *firstblock;
int
unsigned char     **o_p;
{
    int
    struct block_rect *tmpb, *tmpb2;
    struct linablock  *tmpline;

printf("get into line \n");
    img_row = orig_row;
    original_p = o_p;
    firstlineblock->first_block = firstblock->current_block = firstblock;
    make_gap(firstblock);
    if (firstblock->group > 1)
        firstlineblock->group = firstblock->group;
    else firstlineblock->group = 1;
    if (firstblock->attribute&TABLE)
        line_group(firstlineblock, firstblock);
    else if ((firstblock->attribute&(CHAR|FRAME)))
                (((firstblock->attribute&FRAME)))
        line_picture(firstlineblock, firstblock);
    else if (((firstblock->firstsubblock = NULL)){
        line_block(firstlineblock, firstblock, column);
    }
    printf("get into line 1 \n");
    for (num=0,tmpb=firstblock->firstnontextblock; tmpb != NULL; ){
        tmpline = new_contextline(firstlineblock);
        tmpline->right_x = tmpb->right_x;
        tmpline->left_x = tmpb->left_x;
        tmpline->upper_y = tmpb->upper_y;
        tmpline->lower_y = tmpb->lower_y;
        tmpline->width = tmpb->width;
        tmpline->length = tmpb->length;
        tmpline->index = ++num;
        tmpline->att = tmpb->att;
        tmpb2 = tmpb->next;
        remove_nontextblock(tmpb, firstblock);
        tmpb->next = NULL;
        tmpb->previous = NULL;
        tmpb = tmpb2;
        if(((tmpb->attribute&PICTURE)&&((tmpb->att&VOID)))
            &&((tmpb->attribute&GROUP))&&(((firstblock->att&TABLE)))
            ini_line_block(tmpline, tmpb, column, orig_row, o_p);
        else {
            tmpline->first_block = tmpb;
            tmpline->current_block = tmpb;
            }
        tmpb = tmpb2;
    }

/*.....
/*..... Print the line information into a file
/*.....

void print_ln(level, firstline)
int level;
struct linablock *firstline;
{
    int   nextlevel, i;
    struct linablock *tmpsubline,  *tmpline;

    nextlevel = level+1;
    for (tmpsubline = firstline->firstsubline; tmpsubline != NULL;
                                    tmpsubline = tmpsubline->next){
        for (i = 0; i <= (level-1)*8; i++)
            fprintf(line_file, " ");
            fprintf(line_file,"%d index = %d ", tmpsubline, tmpsubline->index);
fprintf(line_file, "x=%d y=%d len=%d wid=%d att=%x under_cap=%d\n",
                        tmpsubline->left_x, tmpsubline->upper_y,
                        tmpsubline->length, tmpsubline->width,
                        tmpsubline->nunsubblock, tmpsubline->att,
                        tmpsubline->under_caption->index);
        }

    for (tmpline = firstline->firstnontextline; tmpline != NULL;
```

【図96】

```
                                                    tmpline=tmpline->next){
            for ( i = 0; i <= (level-1)*8; i++)
                fprintf(line_file, " ");
            fprintf(line_file, "%d nontext index = %d ", tmpline, tmpline->index);
            fprintf(line_file,
  "x=%d y=%d l=%d w=%d att = %x block index1=%d index2=%d g = %d l = %d r = %d\n",
                    tmpline->left_x, tmpline->upper_y, tmpline->length,
                    tmpline->width, tmpline->att, tmpline->first_block->index,
                tmpline->current_block->index, tmpline->group, tmpline->l_caption,
                tmpline->r_caption);
            print_ln(nextlevel, tmpline);
        }
}


/*************************************************************************/
/******** Open the file for the output of line message           ********/
/*************************************************************************/

void print_lineblock(firstline)
struct lineblock *firstline;
{
        line_file = fopen("line_file.dat", "w");
        print_ln(1, firstline);
        fclose(line_file);
}
```

【図118】

```
                                                    tmpsection = tmpsection->next){
        if ((tmpsection->att&(TITLE|HSOLID|VSOLID|HPICTURE))) continue;
        else
            ini_section_block(tmpsection, tmpsection->firstlineblock, p,
                                                              column);
    }
}


/*************************************************************************/
/******** Print the line information into a file                 ********/
/*************************************************************************/

void print_sec(level, firstsec)
int   level;
struct sectionblock *firstsec;
{
        int     nextlevel, i;
        struct sectionblock *tmpsubsec,  *tmpsec;

        nextlevel = level+1;
        for (tmpsubsec = firstsec->firstsubsection; tmpsubsec != NULL;
                                                    tmpsubsec = tmpsubsec->next){
            for ( i = 0; i <= (level-1)*8; i++)
                printf(" ");
                printf("index = %d ", tmpsubsec->index);
                printf("x=%d y=%d l=%d w=%d att = %x \n",
                        tmpsubsec->left_x, tmpsubsec->upper_y,
                        tmpsubsec->length, tmpsubsec->width, tmpsubsec->att);

        }

        for (tmpsec = firstsec->firstnontextsection; tmpsec != NULL;
                                                    tmpsec=tmpsec->next){
            for ( i = 0; i <= (level-1)*8; i++)
                printf(" ");
            printf("nontext index = %d ", tmpsec->index);
            printf("x=%d y=%d l=%d w=%d att = %x \n",
                    tmpsec->left_x, tmpsec->upper_y, tmpsec->length,
                    tmpsec->width, tmpsec->att);
            print_sec(nextlevel, tmpsec);
        }
}


/*************************************************************************/
/******** Open the file for the output of line message           ********/
/*************************************************************************/

void print_secblock(firstsec)
struct sectionblock *firstsec;
{
        print_sec(1, firstsec);
}
```

【図217】

Source Code for

Mono-Spaced ("Courrier") Segmentation

【図97】

```
/*****************************************************************/
/*                                                               */
/*                      File: blocksection.c                     */
/*                      Author: Shin-Yuan Wang                   */
/*                      Date : 1-18-92                           */
/*                                                               */
/*****************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
#include "blocksection.h"
#include "blockgeneral.h"

float  tot1l;
extern float text_length;
unsigned original_y;
int    min_y, max_y;
struct segment_gap *segment;
int    current_seg = 0;

/*****************************************************************/
/*                                                               */
/*       absolute difference of two integers                     */
/*                                                               */
/*****************************************************************/

int abs(a, b)
int a, b;
{
    return (a>b)?a-b:b-a;
}

/*****************************************************************/
/*                                                               */
/*       allocation lineblockpointer                             */
/*                                                               */
/*****************************************************************/

struct lineblock **new_linepointer(nl, nh)
int nl, nh;
{
    struct lineblock **v;

    v = (struct lineblock **) malloc((nh-nl+1)*sizeof(struct lineblock*));
    if (!v) {
        printf("error: in the allocation of linepointer\n");
        exit(1);
    }
    v = v-nl;
    return v;
}

/*****************************************************************/
/*                                                               */
/*       allocation section lineblockpointer                     */
/*                                                               */
/*****************************************************************/

struct sectionblock **new_sectionpointer(nl, nh)
int nl, nh;
{
    struct sectionblock **v;

    v = (struct sectionblock **) malloc((nh-nl+1)*sizeof(struct sectionblock*));
    if (!v) {
        printf("error: in the allocation of sectionpointer\n");
        exit(1);
    }
    v = v-nl;
    return v;
}

/*****************************************************************/
/*                                                               */
/*       sort line block pointer array                           */
/*                                                               */
/*****************************************************************/

void sort_linepointer(line, num)
struct lineblock **line;
int    num;
{
    int    i, j, stop;
    struct lineblock *tmp;

    for (i = 0; i < num-1 ; i++){
        for (j = 0, stop = 0; j < num -1 -i; j++)
            if (line[j]->upper_y > line[j+1]->upper_y){
                tmp  = line[j+1];
                line[j+1] = line[j];
                line[j] = tmp;
                stop = 1;
            }
        if (stop == 0) break;
    }
}

/*****************************************************************/
/*                                                               */
/*       line block sorting                                      */
/*                                                               */
/*****************************************************************/

struct lineblock **sort_y(firstlineblock,  numline)
struct lineblock *firstlineblock;
int    *numline;
{
    struct lineblock  *tmp;
    struct lineblock **line_x;
    int    i, j, stop;

    *numline = 0;
    for ( tmp = firstlineblock->firstentities; tmp != NULL; tmp = tmp->next)
        (*numline)++;
    for ( tmp = firstlineblock->firstcontextline; tmp != NULL; tmp = tmp->next)
        (*numline)++;

    line_x = new_linepointer(0, (*numline)-1);

/*****************************************************************/
/*                                                               */
/*       allocation new segment_gap structure                    */
/*                                                               */
/*****************************************************************/

struct segment_gap *new_segment(nl, nh)
int nl, nh;
{
    struct segment_gap *v;
    int    i;

    v = (struct segment_gap *) malloc((nh-nl+1)*sizeof(struct segment_gap));
    if (!v) {
        printf("error: in the allocation of segment_gap\n");
        exit(1);
    }
    v = v-nl;
```

【図98】

```
    for (i = 0, tmp = firstlineblock->firstsubline; tmp != NULL; tmp = tmp->next)
        line_x[i++] = tmp;

    /* extra testing */

    if ((firstlineblock->currentsubline != NULL)&&
        (i != firstlineblock->currentsubline->index))
        printf("error: in sort_y\n");

    for (tmp = firstlineblock->firstcontextline; tmp != NULL; tmp = tmp->next)
        line_x[i++] = tmp;

    sort_linepointer(line_x, numline);
    return line_x;
}

/*****************************************************************************/
/********* section block sorting                                             */
/*****************************************************************************/

struct sectionblock **sort_section(first, num)
struct sectionblock *first;
int *num;
{
    struct sectionblock *tmp;
    struct sectionblock **line_x;
    int   i, j, step;

    *num = 0;
    for (tmp = first->firstsubsection; tmp != NULL; tmp = tmp->next)
        (*num)++;

    line_x = new_sectionpointer(0, (*num)-1);

    for (i = 0, tmp = first->firstsubsection; tmp != NULL; tmp = tmp->next)
        line_x[i++] = tmp;

    for ( i = 0; i < (*num) -1; i++){
        for (j = 0, step = 0; j < (*num)-i-1; j++)
            if (((line_x[j]->upper_y > line_x[j+1]->upper_y) ||
                 (((line_x[j]->upper_y == line_x[j+1]->upper_y)&&
                   (line_x[j]->lower_y > line_x[j+1]->lower_y))) {
                tmp  = line_x[j+1];
                line_x[j+1] = line_x[j];
                line_x[j] = tmp;
                step = 1;
            }
        if (step == 0) break;
    }
    return line_x;
}

/*****************************************************************************/
/********* Allocate new line block                                           */
/*****************************************************************************/

struct sectionblock *new_secblock(ni, nh)
int ni, nh;
{
        int i;
        struct sectionblock *v;

        v = (struct sectionblock *) malloc((unsigned) (nh-ni+1)*
                                           sizeof(struct sectionblock));
        if (!v){
            printf("allocation error: section block \n");
            exit(1);
        }
        v -= ni;
        for (i = ni; i <= nh; i++) {
            v[i].firstlineblock = v[i].currentlineblock = NULL;
            v[i].firstsubsection = v[i].currentsubsection = NULL;
            v[i].firstcontextsection = v[i].currentcontextsection = NULL;
            v[i].previous = v[i].next = NULL;
            v[i].upper_y = v[i].lower_y = v[i].left_x = v[i].right_x = 0;
            v[i].width = v[i].length = 0;
            v[i].att = UNKNOWN;
            v[i].mainsecblock = 0;
            v[i].index = 0;
            v[i].boundary_index = -1;
        }
        return v;
}

/*****************************************************************************/
/********* chain the context section block                                   */
/*****************************************************************************/

struct sectionblock *new_ntsection(headsection)
struct sectionblock *headsection;
{
    struct sectionblock *n;

    n = new_secblock(0,0);
    if (headsection->firstcontextsection == NULL)
        headsection->firstcontextsection = n;
    else{
        headsection->currentcontextsection->next = n;
        n->previous = headsection->currentcontextsection;
    }
    headsection->currentcontextsection = n;
    return n;
}

/*****************************************************************************/
/********* chain the section block                                           */
/*****************************************************************************/

struct sectionblock *new_section(headsection)
struct sectionblock *headsection;
{
    struct sectionblock *n;

    n = new_secblock(0,0);
    if (headsection->firstsubsection == NULL)
        headsection->firstsubsection = n;
    else{
        headsection->currentsubsection->next = n;
        n->previous = headsection->currentsubsection;
    }
    headsection->currentsubsection = n;
    return n;
}

/*****************************************************************************/
/********* take linablock from section                                       */
/*****************************************************************************/

void take_section(line, currsection)
struct linablock    *line;
struct sectionblock *currsection;
```

【図99】

```
{
    if (currsection->firstlinablock == line){
        currsection->firstlinablock = line->next;
        if (currsection->currentlinablock == line)
            currsection->currentlinablock = NULL;
        else
            currsection->firstlinablock->previous = NULL;
    } else if (currsection->currentlinablock == line){
            currsection->currentlinablock = line->previous;
            line->previous->next = NULL;
        } else {
                line->previous->next = line->next;
                line->next->previous = line->previous;
            }
    line->previous = NULL;
    line->next = NULL;
}
/*.....*/
/*.....  put linablock into section                    .....*/
/*.....                                                .....*/

void put_section(line, currsection)
struct linablock  *line;
struct sectionblock *currsection;
{
    if (currsection->firstlinablock == NULL){
        currsection->firstlinablock = line;
        currsection->currentlinablock = line;
        line->previous = NULL;
        line->next = NULL;
    } else {
            currsection->currentlinablock->next = line;
            line->previous = currsection->currentlinablock;
            currsection->currentlinablock = line;
            line->next = NULL;
        }
}
/*.....*/
/*.....  create new samaline structure                 .....*/
/*.....                                                .....*/

struct sama_line *new_samaline(curr)
struct linablock *curr;
{
    struct samaline *v;

    v = (struct samaline *) malloc(sizeof(struct samaline));
    if (!v) {  printf("allocation error: in samaline\n");
               exit(1);
            }
    v->current = curr;
    v->previous = NULL;
    v->next = NULL;
    v->r_connection = v->l_connection = NULL;
    return v;
}
/*.....*/
/*.....  sort samaline array by x                      .....*/
/*.....                                                .....*/

void sort_samline(first)
struct samaline *first;
{
    int   stop, tmpindex;
    struct samaline *tmp;
    struct linablock *tmpline;

    for ( ; ; ){
        stop = 0;
        for (tmp = first->next; tmp != NULL; tmp = tmp->next)
            if (tmp->current->left_x < tmp->previous->current->left_x){
                tmpline = tmp->current;
                tmpindex = tmp->index;
                tmp->current = tmp->previous->current;
                tmp->index = tmp->previous->index;
                tmp->previous->current = tmpline;
                tmp->previous->index = tmpindex;
                stop = 1;
            }
        if (stop == 0) break;
    }
}
/*.....*/
/*.....  remove linablock                              .....*/
/*.....                                                .....*/

void line_rm(test, headline, line_rm)
int test;
struct linablock *headline, *line_rm;
{
    if (test == TRUE){
        if (headline->firstsubline == line_rm){
            headline->firstsubline = line_rm->next;
            if (line_rm->next != NULL) line_rm->next->previous = NULL;
            else headline->currentsubline = NULL;
        } else if (headline->currentsubline == line_rm){
                headline->currentsubline = line_rm->previous;
                line_rm->previous->next = NULL;
            } else {
                line_rm->next->previous = line_rm->previous;
                line_rm->previous->next = line_rm->next;
            }
    } else {
        if (headline->firstnontextline == line_rm){
            headline->firstnontextline = line_rm->next;
            if (headline->firstnontextline != NULL) line_rm->next->previous = NULL;
            else if (headline->currentnontextline == NULL);
        } else if (headline->currentnontextline == line_rm){
                headline->currentnontextline = line_rm->previous;
                line_rm->previous->next = NULL;
            } else {
                line_rm->next->previous = line_rm->previous;
                line_rm->previous->next = line_rm->next;
            }
    }
}
/*.....*/
/*.....  combination of line block  forward            .....*/
/*.....                                                .....*/

void fline_combine(newhead, line_rmn)
struct linablock *newhead, *line_rmn;
{
```

【図100】

```
    newhead->first_block->previous = line_tran->current_block;
    line_tran->first_block->next = newhead->first_block;
    newhead->first_block = line_tran->first_block;
    newhead->numunblock += line_tran->numunblock;
    if (line_tran->left_x < newhead->left_x)
                        newhead->left_x = line_tran->left_x;
    if (line_tran->right_x > newhead->right_x)
                        newhead->right_x = line_tran->right_x;
    if (line_tran->upper_y < newhead->upper_y)
                        newhead->upper_y = line_tran->upper_y;
    if (line_tran->lower_y > newhead->lower_y)
                        newhead->lower_y = line_tran->lower_y;
    newhead->width = newhead->right_x-newhead->left_x+1;
    newhead->length = newhead->lower_y-newhead->upper_y+1;
}

/*.....................................................................*/
/*.....                                                           .....*/
/*.....  combination of the block backward                        .....*/
/*.....                                                           .....*/
/*.....................................................................*/

void blne_combine(newhead, line_tran)
struct linablock *newhead, *line_tran;
{

    newhead->current_block->next = line_tran->first_block;
    line_tran->first_block->previous = newhead->current_block;
    newhead->current_block = line_tran->current_block;
    newhead->numunblock += line_tran->numunblock;
    if (line_tran->left_x < newhead->left_x)
                        newhead->left_x = line_tran->left_x;
    if (line_tran->right_x > newhead->right_x)
                        newhead->right_x = line_tran->right_x;
    if (line_tran->upper_y < newhead->upper_y)
                        newhead->upper_y = line_tran->upper_y;
    if (line_tran->lower_y > newhead->lower_y)
                        newhead->lower_y = line_tran->lower_y;
    newhead->width = newhead->right_x-newhead->left_x+1;
    newhead->length = newhead->lower_y-newhead->upper_y+1;
}

/*.....................................................................*/
/*.....  line remove from one head and merge with another head  .....*/
/*.....................................................................*/

void line_trans(*line_tran, originalhead, newhead, att)
struct linablock *line_tran, *originalhead, *newhead;
int att;
{

    line_print(, originalhead, line_tran);
    blne_combine(newhead, line_tran);
    free((struct linablock*) line_tran);
}

/*.....................................................................*/
/*.....                                                           .....*/
/*.....  test if one line inside another line                     .....*/
/*.....                                                           .....*/
/*.....................................................................*/

int if_inside(firstline, secondline)
struct linablock *firstline, *secondline;
{

    if (((float)cover_range(firstline->upper_y, firstline->lower_y,
                secondline->upper_y, secondline->lower_y) > 1.0) &&
        ((float)cover_range(firstline->left_x, firstline->right_x, 1.0)
                secondline->left_x, secondline->right_x) >= 1.0))
        return 1;
    return 0;
}

/*.....................................................................*/
/*.....                                                           .....*/
/*.....                                                           .....*/
/*.....                                                           .....*/
/*.....................................................................*/

float cover_mark(mark, a1, a2)
int *mark;
int a1, a2;
{
    int unmark, i;

    for ( i = a1, unmark = 0; i <= a2; i++){
        if (mark[i] == 0){
            unmark++;
            mark[i] = 1;
        }
    }
    return ((float)unmark/((float)(a2-a1+1)));
}

/*.....................................................................*/
/*.....                                                           .....*/
/*.....  the number of 'neighbor' on the left side               .....*/
/*.....                                                           .....*/
/*.....................................................................*/

void test_left_neighbor(line, leftlevel, leftlength, head, linelist, nodistroy)
struct linaline *line;
int  *leftlevel;
float  *leftlength;
struct linablock *head;
struct linablock **linelist;
struct linaline *nodistroy;
{
    int  n, upper, lower, *mark, nostleft;
    struct linaline *maxline, *minline, *tmp;

    UPPER = line->current->upper_y;
    lower = line->current->lower_y;
    for ( tmp = line->current->left; tmp != NULL; tmp = tmp->previous){
        if (tmp->current == nodistroy) continue;
        if (tmp->current->upper_y < UPPER) upper = tmp->current->upper_y;
        if (tmp->current->lower_y > lower) lower = tmp->current->lower_y;
    }

    mark = (int *)ivector(upper, lower);
    nostleft = line->current->left_x;
    maxline = minline = NULL;
    *leftlevel = 0;
    *leftlength = 0.0;
    for ( tmp = line->previous; tmp != NULL; tmp=tmp->previous){
        if (tmp->current == NULL) continue;
        if ((tmp->current->right_x>nostleft-line->current->length)&&
            (cover_range(tmp->current->upper_y, tmp->current->lower_y,
             line->current->upper_y, line->current->lower_y)>0.0)) {
            if ((n = cover_mark(mark,tmp->current->upper_y,tmp->current->lower_y)
                                                        > OVER_LAP)
                ){
                if (tmp->current->left_x < nostleft)
                    nostleft = tmp->current->left_x;
                *leftlength += (float)tmp->current->length*n;
                if (n != 0.0) {
                    (*leftlevel)++;
                    if (*linelist == NULL)*linelist = tmp;
                    maxline = tmp;
                    minline = tmp;
```

【図１０１】

```
    } else if (tmp->current->length >= maxline->current->length)
        maxline = tmp;
      else
        if (tmp->current->length < minline->current->length)
          minline = tmp;
    }
  }
}

if (leftlevel == 2){
  if (minline->current->width < minline->current->right_x){
    maxline->current->left_x, minline->current->right_x)){
    if ((minline->current->width < minline->current->length*0)&&
        (leftlength < ((float)line->current->length*1.5))&&
        ((minline->current->length >= minline->current->length*2))||
        (maxline->current->width < maxline->current->length * 2))){

      -leftlevel = 1;

      if (modistroy ) minline){
        line_transfer(minline->current.head,maxline->current,
                      minline->current->att);
        maxline->current = NULL;
        lineslist[minline->index] = NULL;
        minline = NULL;
      }
      else {
        line_transfer(maxline->current,head,minline->current,
                      maxline->current->att);
        maxline->current = NULL;
        lineslist[maxline->index] = NULL;
        maxline = NULL;
      }
    } else {
      leftlevel = 1;
      if (minline->current->left_x < maxline->current->left_x) {
        leftlength = maxline->current->length;
        minline  = NULL;
      } else {
        leftlength = minline->current->length;
        maxline = NULL;
      }
    }
  }
  if (leftlevel == 1)
    line->_connection = (maxline==NULL)?minline:maxline;

  if((leftlevel >= 2) *leftlength /= (float)*leftlevel;
  free((int *) (mark.upper));
}

/*.....*/
/*.....*/
/*.....*/

void test_right_neighbor(line, rightlength, head, lineslist, modistroy)
struct nameline *line;
int
float
struct linesblock *head, **lineslist;
struct nameline *modistroy;
{
  int   upper, lower, *mark, mostright;
  struct nameline *maxline, *minline, *tmp;
  float m;

  upper = line->current->upper_y;
  lower = line->current->lower_y;
  for (tmp = line->next; tmp != NULL; tmp = tmp->next){
    if (tmp->current == NULL) continue;
    if ((tmp->current->upper_y < upper) upper = tmp->current->upper_y;
    if (tmp->current->lower_y > lower) lower = tmp->current->lower_y;
  }

  mark = (int *)ivector(upper, lower);
  mostright = line->current->right_x;
  maxline = minline = NULL;
  *rightlevel = 0;
  *rightlength = 0.0;
  for ( tmp = line->next; tmp != NULL; tmp=tmp->next){
    if (tmp->current == NULL) continue;
    if (!(in_inside(tmp->current, line->current))){
      line_transfer(tmp->current, head, line->current, tmp->current->att);
      tmp->current = NULL;
      lineslist[tmp->index] = NULL;
      continue;
    }
    if ((tmp->current->left_x<=mostright+line->current->length)&&
        (cover_range(tmp->current->upper_y, tmp->current->lower_y,
                     line->current->upper_y, line->current->lower_y) > 0.0)){
      m = cover_mark(mark,tmp->current->upper_y,tmp->current->lower_y);
      printf("m = %f \n", m);
      if ((tmp->current->right_x <= mostright+1)||(m >= OVERLAP))
        {
          if (tmp->current->right_x > mostright)
            mostright = tmp->current->right_x;
          *rightlength += (float)tmp->current->length;
          if (m != 0.0) {
            (*rightlevel)++;

            if (maxline == NULL||(minline == NULL)){
              maxline = tmp;
              minline = tmp;
            } else if (tmp->current->length >= maxline->current->length)
                maxline = tmp;
              else if (tmp->current->length < minline->current->length)
                minline = tmp;
          }
        }
  }
  printf("right level %d \n", *rightlevel);
  if (*rightlevel == 2){
    if (if_overlap(minline->current->left_x, minline->current->right_x,
                   maxline->current->left_x, maxline->current->right_x)){
      if ((minline->current->width < minline->current->length*0)&&
          (*rightlength < ((float)line->current->length*1.5))&&
          ((maxline->current->length >= minline->current->length*2))||
          (maxline->current->width < maxline->current->length * 2))
        {
          *rightlevel = 1;
          if (modistroy != minline){
            line_transfer(minline->current.head,maxline->current,
                          minline->current->att);
            lineslist[minline->index] = NULL;
            minline = NULL;
```

【図102】

```
                    line_transfer(maxline->current.head,minline->current,
                                  maxline->current->att);
                    maxline->current = NULL;
                    linelist[maxline->index] = NULL;
                    maxline = NULL;
                }
            } else {
                if (rightlevel > 1){
                    minline->current->left_x < maxline->current->left_x) {
                    *rightlength = minline->current->length;
                    maxline = NULL;
                } else {
                    *rightlength = maxline->current->length;
                    minline = NULL;
                }
            }
        }

    if (*rightlevel == 1)
        line->r_connection = (maxline==NULL)?minline:maxline;

    if(*rightlevel >= 2) *rightlength /= (float)*rightlevel;

    free((int *) (mark+upper));
}

/*.....text_left_neighbor(line,leftlevel,leftlen,first,linelist, nodist);
/*.....text_right_neighbor(line, rightlevel, rightlen, first, linelist, nodist);
/*.....                                                          .........*/

void text_neighbor(line,leftlevel,rightlevel,leftlen,rightlen,first,linelist, nodist)
struct sameline *line;
int   *leftlevel, *rightlevel;
float *leftlen, *rightlen;
struct lineblock *first;
struct lineblock **linelist;
struct sameline *nodist;
{
    text_left_neighbor(line, leftlevel, leftlen, first, linelist, nodist);
    text_right_neighbor(line, rightlevel, rightlen, first, linelist, nodist);
}

/*.....Make title toward left direction                          .........*/

void left_title(currentline, line, first)
struct sameline *currentline;
struct lineblock **line;
struct lineblock *first;
{
    struct sameline *tmpline;
    int   leftmost, leftlevel, rightlevel;
    float leftlength, rightlength;

    leftmost = currentline->current->left_x;
    for ( tmpline = currentline->l_connection; tmpline != NULL;
                tmpline = tmpline->r_connection){
        if (tmpline->current == NULL) continue;
printf("left = %d \n", tmpline->current->index);
        if (tmpline->current->left_x < leftmost->current->left_x) {
            leftmost = tmpline->current;
            *leftlength = tmpline->current->length;
            *leftlevel = 1;
        } else if (leftmost->current->left_x == leftmost->current->left_x) {
            if (*rightlength < tmpline->current->length) {
                leftmost = tmpline->current;
                *leftlength = tmpline->current->length;
                *leftlevel = 1;
            } else if (*rightlength == tmpline->current->length) {
                *leftlevel = *leftlevel + 1;
            }
        }
    }
    if (leftmost == currentline->l_connection) return;
printf("left = %d \n", tmpline = currentline->l_connection){
    if (tmpline->current == NULL) continue;
    if (tmpline->current->index==index)
        if (tmpline->right_x >= leftmost->tmpline->current->length){

text_right_neighbor(tmpline, &rightlevel, &rightlength, first, line, curr
    if ((rightlevel == 1)&&(tmpline->r_connection == currentline)){
        if (cover_range(tmpline->current->upper_y,
                        tmpline->current->lower_y, currentline->current->upper_y,
                        currentline->current->lower_y)>=0.3){
            if (tmpline->current->left_x < leftmost)
                leftmost = tmpline->current->left_x;
            line_transfer(tmpline->current, first, currentline->current,
                          tmpline->current->tail);
            tmpline->current = NULL;
            currentline->att = TITLE|TEXT;
            line[tmpline->index] = NULL;
            text_left_neighbor(currentline,&leftlevel,&leftlength,first,line, NULL
                               );
            if (leftlevel == 1)
                continue;
            else
                break;
        } else {
            tmpline->r_connection = NULL;
            currentline->current->att = TEXT;
            break;
        }
    } else {
        tmpline->r_connection = NULL;
        break;
    }
} else break;
}
currentline->l_connection = NULL;

/*.....Make title toward right direction                         .........*/

int right_title(currentline, line, first)
struct sameline *currentline;
struct lineblock **line;
struct lineblock *first;
{
    struct sameline *tmpline;
    int   rightmost, rightlevel, leftlevel;
    float rightlength, leftlength;
    int   combine;

combine = 1;
rightmost = currentline->current->right_x;
for ( tmpline = currentline->r_connection; tmpline != NULL;
                tmpline = currentline->r_connection){
    if (tmpline->current == NULL) continue;
printf("tmpline %d currentline %d\n", tmpline->current->index,
                currentline->current->index );

    if (tmpline->current->left_x <= rightmost->tmpline->current->length){
        text_left_neighbor(tmpline, &leftlevel, &leftlength, first, line, current
line);
printf("tmpline->l_co %d leftlevel = %d \n",
                tmpline->l_connection->current->index, leftlevel);
    if ((leftlevel == 1)&&(tmpline->l_connection == currentline)){
    if (cover_range(tmpline->current->upper_y,
                    tmpline->current->lower_y, currentline->current->upper_y,
                    currentline->current->lower_y) >= 0.5) {
```

【図103】

```
    if (tmpline->current->right_x > rightmost)
        rightmost = tmpline->current->right_x;
    line_transfer(tmpline->current, first, currentline->current,
                                        tmpline->current->att);

    currentline->current->att = TEXT|TEXT;
    tmpline->current = NULL;
    line[tmpline->index] = NULL;
    text_right_neighbor(currentline, &rightlevel, &rightlength,
                                        first, line, NULL);
    if (rightlevel == 1)
        continue;
    else
        break;
    } else {
        tmpline->l_connection = NULL;
        currentline->current->att = TEXT;
        break;
    }
} else {
    tmpline->l_connection = NULL;
    break;
    }
} else break;
}

/*.....*/
/*..... Pack a set of line block                    .....*/
/*.....                                             .....*/

void pack_lines(line, num)
struct lineblock **line;
int
*num;
{
    int     i, j, pack;
    struct lineblock **tmpline;

    tmpline = new_linepointer(0, (*num)-1);
    pack = 0;
    for ( i = 0, j = 0; i < *num; i++){
        if (line[i] == NULL)
            pack = 1;
        else
            tmpline[j++] = line[i];
    }
    if (pack == 1) {
        for ( i = 0; i < j; i++) line[i] = tmpline[i];
        for ( i = j; i < *num; i++) line[i] = NULL;
        *num = j;
    }
    free((struct linepointer **) tmpline);
}

/*.....*/
/*..... index for combining two sameline  block     .....*/
/*.....                                             .....*/

int index_of_combine(line1, line2)
struct lineblock *line1, *line2;
{

    float i, m, p, q;

    i = cover_range(line1->upper_y, line1->lower_y, line2->upper_y,
                    line2->lower_y);
    m = cover_range(line1->left_x, line1->right_x, line2->left_x,
                    line2->right_x);
    p = cover_range(line1->left_x, line1->right_x, line2->left_x,
                    line1->right_x);
    q = cover_range(line2->left_x, line2->right_x, line1->left_x,
                    line1->right_x);
    if ((line1->att != TEXT)&&(line2->att != TEXT) {
        if ((i >= 0.5)||(m >= 0.5)) return 1;
        else return 0;
    }
    if ((line1->att == TEXT)&&(line2->att != TEXT)) {
        if (i >= 0.0) return 0;
        if (m < 1.0/3.0) return 2;
        else return 1;
    }
    if ((line1->att != TEXT)&&(line2->att == TEXT)) {
        if (i >= 0.0) return 0;
        if (m < 1.0/3.0) return 2;
        else return 1;
    }
    if ((line1->att == TEXT)&&(line2->att == TEXT)){
        if ((i >= 0.5) || (m >= 0.5)) return 1;
        else return 0;
    }
    /*
    if (i <= m) {
        if ((i < 1.0/3.0)&&(line2->att != TEXT) return 0;
        else return 1;
    } else {
        if ((m < 1.0/3.0)&&(line1->att != TEXT) return 0;
        else return 1;
    }
    */
}

/*.....*/
/*..... Combine touched line element inside  the sameline  .....*/
/*.....                                             .....*/

void inside_test(first, head, linelist)
struct sameline *first;
struct lineblock *head;
struct lineblock **linelist;
{
    int     m;
    struct sameline *tmp, *tmp1;

printf("before inside test\n");
for ( tmp = first; tmp != NULL; tmp = tmp->next){
    if (tmp->current == NULL) printf("null\n");
    else printf("index = %d x = %d y = %d l = %d v = %d \n",
                tmp->current->index, tmp->current->left_x,
                tmp->current->upper_y, tmp->current->length,
                tmp->current->width);
}

for (tmp = first; tmp != NULL; tmp = tmp->next){
    if (tmp->current == NULL) continue;
    for (tmp1 = tmp->next; tmp1 != NULL; tmp1 = tmp1->next){
        if (tmp1->current == NULL) continue;
        if (tmp1->current->left_x > tmp->current->right_x) break;
printf("index 2 = %d \n", tmp->current->index, tmp1->current->index);
        if ((m = index_of_combine(tmp->current, tmp1->current)) == 1){
            if ((tmp->current->att == TEXT) ||
                ((tmp->current->att != TEXT)&&(tmp1->current->att != TEXT))){
```

【図104】

```
line_transfer(tmp->current, head, tmp1->current,
                        tmp->current->att);

    tmp->current = NULL;
    linelist[tmp->index] = NULL;
    if (!(tmp1->current->att&TEXT))
      tmp1->current->att = (TITLE|TEXT);
  } else if (tmp1->current->att & (TITLE|TEXT)) {
    line_transfer(tmp1->current, head, tmp->current,
                  tmp1->current->att);

    tmp1->current = NULL;
    linelist[tmp1->index] = NULL;
    if (!(tmp->current->att&TEXT))
      tmp->current->att = (TITLE|TEXT);
    break;
  }
} else if (a == 2) {

  if (tmp->current->att != TEXT) {
    line_transfer(tmp1->current, head, tmp->current,
                  tmp1->current->att);

    tmp1->current = NULL;
    linelist[tmp1->index] = NULL;
  } else {
    line_transfer(tmp->current, head, tmp1->current,
                  tmp->current->att);

    tmp->current = NULL;
    linelist[tmp->index] = NULL;
    break;
  }
}
}
}

printf("\n\n after inside test\n");
for ( tmp = first; tmp != NULL; tmp = tmp->next){
  if (tmp->current == NULL) printf("NULL\n");
  else printf("index = %d x = %d y = %d v = %d \n",
              tmp->current->index, tmp->current->x,
              tmp->current->upper_y, tmp->current->length,
              tmp->current->width);
}
}

/*****************************************************************/
/*                                                               */
/*********** formation of title                                  */
/*                                                               */
/*****************************************************************/

void title_scan(line_y_sort, firstline, num)
struct linelock **line_y_sort;
struct linelock *firstline;
int   *num;
{
    int   i, j, k;
    int   leftlevel, rightlevel;
    int   top, bottom, rightlength;
    float leftlength;
    struct linelock *tmp;
    struct samaline *first, *currentline, *saveline;

printf("get into title_scan\n");
    for ( i = 0, i < *num; i++){
      if (line_y_sort[i] == NULL) continue;
      if (line_y_sort[i]->att != UNKNOWN) continue;
      first = new_samaline(line_y_sort[i]);
      first->index = i;
      currentline = first;
      top = line_y_sort[i]->upper_y;
      tmp = line_y_sort[i]->next;
      bottom = line_y_sort[i]->lower_y;
      for ( j = i+1; j < *num; j++){
        if (line_y_sort[j] == NULL) continue;
        if (line_y_sort[j]->lower_y < top) break;
        if (line_y_sort[j]->upper_y > bottom) break;
        if (!(leftlevel >= 2)&&(rightlevel>=2)){
          currentline->att = (TEXT|PICTURE|RECTEN);
        }
        if (leftlevel == 1){
          if ((int)((float)currentline->length/leftlength) > 1){
            currentline->current->att = (PICTURE|RECTEN|TEXT);
          } else {
            left_title(currentline, line_y_sort, firstline);
            test_right_neighber(currentline, &rightlevel,
                                &rightlength, firstline, line_y_sort, NULL);
          }
          if (rightlevel == 1){
```

【図105】

```
    if ((int)((float)currentline->current->length/rightlength) > 1){
        currentline->current->att = PICTURE|RESTEST|TEXT2;
    } else right_title(currentline, line_y_sort, firstline);
  }
  if (((leftlevel >= 2)&&(rightlevel == 0))||
      ((leftlevel == 0)&&(rightlevel >= 2))){
    currentline->current->att = TEXT|PICTURE|RESTEST;
  }
  if ((leftlevel == 0)&&(rightlevel == 0)) {
    currentline->current->att = PICTURE|TEXT|RESTEST;
  }
  if (currentline->current->att == UNKNOWN)
    currentline->current->att = PICTURE|TEXT|RESTEST;

  currentline->current = NULL;
 }
}
pack_line(line_y_sort, num);
sort_linepointer(line_y_sort, *num);
}

/*.....formation of text boundary.....*/

struct lineblock **boundary_form(line_x, num_bound, num)
struct lineblock **line_x;
*num_bound, *num;
int
{
    int    i, j;
    struct lineblock *topline;
    struct lineblock **bound;

    for (i=0; i<*num_bound - 0; i < *num; i++)
        if (line_x[i]->attA&RESTEXT) (*num_bound)++;
/* printf("numbound = %d\n", *num_bound); */
    if ((*num_bound) == 0) return NULL;

    bound = new_linepointer(0, (*num_bound)-1);
    for (i = 0, j = 0; i < *num; i++)
    {
        if (line_x[i]->attA&RESTEXT){
            bound[j++] = line_x[i];
            line_x[i] = NULL;
        }
    }
    sort_linepointer(bound, *num_bound);

    pack_line(line_x, num);
    return bound;
}

/*.....combine two line.....*/

void linecombine(line1, line2)
struct lineblock *line1, *line2;
{

    if (line1->left_x <= line2->left_x)
        line_combine(line1, line2);
    else
        line_combine(line2, line1);
```

```
/*.....formation of section block.....*/

struct sectionblock *form_section(headsection, first, last, list_y_sort, column, this
)
struct sectionblock *headsection;
int    first, last;
struct lineblock **list_y_sort;
int    column, thre;
{

    int    i,j, k, p;
    int    columnset, numsection, level;
    int    *hist_column;
    int    gapbegin[GAP], gapend[GAP];
    int    th;
    float  allmgap;
    float  gap[GAP];
    struct sectionblock *nsection, *section, *currentsection;
    struct lineblock    *tmp, *topline, *appline;

    printf("get into form section\n");
    hist_column = (int *)malloc(column-1);
    for (i = 0; i < column; hist_column[i++] = 0);

    for (i = first; i <= last; i++){
        if (list_y_sort[i] == NULL) continue;
        for (j = list_y_sort[i]->left_x; j <= list_y_sort[i]->right_x; j++)
            hist_column[j]++;
    }

    for ( i = 0; i < GAP; i++){
        gapbegin[i]=gapend[i]= 0;
        gap[i] = 0.0;
    }

    /* column estimation */

/*  th = (float)(list_y_sort[last]->lower_y-list_y_sort[first]->upper_y)/
                                        (float)(last*4.0);
*/
    th = (float)th;
/*  th = 0.0;
*/
/*
printf(" first = %d last = %d th = %f\n", list_y_sort[first]->index,
                list_y_sort[last]->index, th);
for (i = 0; i < column; i++)
    if (hist_column[i] != 0){
        printf("start from i = %d \n", i);
        for (j = i; j < column; j++)
            printf("%d ", hist_column[j]);
        break;
    }
*/
    columnset = -1;
```

【図106】

```
for ( i = 0; i < column; i){
    if ((float)hist_column[i] == thi(
        gapbegin[i+columnest] = i;
        gap[columnest] = 0.0;
        for ( j = i; j < column)&&((float)hist_column[j] <= th);
            gapend[columnest] = (float)hist_column[j]; j++); /*
        gap[columnest] /= ((float)(gapend[columnest]-gapbegin[columnest]+1);
        for ( j = gapbegin[columnest], level = hist_column[j];
                j < gapend[columnest]; j++)
            if (((float)hist_column[j] > gap[columnest]){
                if (hist_column[j] <= level) {
                    level = hist_column[j];
                    continue;
                } else break;
            } else break;
        gapbegin[columnest] = j;
        for ( j = gapend[columnest], level = hist_column[j];
                j->gapbegin[columnest] ; j--)
            if (((float)hist_column[j] > gap[columnest]){
                if (hist_column[j] <= level){
                    level = hist_column[j];
                    continue; hist_column[j]);
                } else break;
            } else break;
        gapend[columnest] = j;
        if ((gapend[columnest]-gapbegin[columnest]) <= 1)
            columnest--;
        else if ((columnest >= 1)&&
                ((gapbegin[columnest-1]-1)>(int)text*2))){
            if ((gapend[columnest]-gapbegin[columnest])>
                (gapend[columnest-1]-gapbegin[columnest-1])){
                gapbegin[columnest-1] = gapbegin[columnest];
                gapend[columnest-1] = gapend[columnest];
            }
            columnest--;
        }

    if (columnest == CAP-1) break;
    else continue;
  }
  else
    i++;
}

/* group section */

nsection = new_sectblock(0, columnest);

for ( i = first; i <= last; i++){
    if (list_y_sort[i] == NULL) continue;
    for (k = 0, j = 0; k < columnest; k++)
        if ((list_y_sort[i]->left_x > gapend[k]) &&
            (list_y_sort[i]->right_x < gapbegin[k+1])){
            put_section(list_y_sort[i], &(nsection[k]));
            j = 1;
            break;
        }
    if (j == 0)
        put_section(list_y_sort[i], &(nsection[columnest]));
  }

if (columnest > 0){
segment[current_seg].column_no = columnest;
segment[current_seg].begin_column = ivector(1, columnest);
segment[current_seg].end_column = ivector(1, columnest);
for ( i = 1, j = columnest; i++){
    segment[current_seg].begin_column[i] = gapend[i-1];
    segment[current_seg].end_column[i] = gapbegin[i];
}

for ( j = 1, j <= segment[current_seg].column_no; j++)
    printf("bg = %d end = %d ", segment[current_seg].begin_column[j],
                                segment[current_seg].end_column[j]);

  printf("\n");
}

for (tapline = nsection[columnest].firstlineblock; tapline != NULL;
                                    tapline = tapline->next){
    for ( k = 0, k < columnest; k++)
        if (tapline->left_x < gapbegin[k+1]&&
            (tapline->left_x > gapend[k]))
            break;
    if (k != columnest) {
        for (tap = nsection[k].firstlineblock; tap != NULL; ){
            if ((cover_range(tap->upper_y, tap->lower_y, tapline->upper_y,
                (cover_range(tap->upper_y, tap->lower_y, tapline->lower_y,
                tapline->lower_y) > OVERLAP) || (cover_range(tapline->upper_y,
                    tap->lower_y) > OVERLAP)){
                tapline = tap->next;
                take_section(tap, &(nsection[k]));
                if (tap->right_x > tapline->left_x - text)
                    line_combine(tapline, tap);
                else
                    put_section(tap, &(nsection[columnest]));
                tap = tapline;
                continue;
            }
            tap = tap->next;
        }
    }
}

for ( k = 0; k < columnest; k++)
    if ((tapline->right_x < gapbegin[k+1])&&
        (tapline->right_x > gapend[k]))
        break;

if (k != columnest) {
    for (tap = nsection[k].firstlineblock; tap != NULL; ){
        if ((cover_range(tap->upper_y, tap->lower_y, tapline->upper_y,
            tapline->lower_y) > OVERLAP) || (cover_range(tapline->upper_y,
                tap->lower_y) > OVERLAP)){
            tapline = tap->next;
            take_section(tap, &(nsection[k]));
            if (tap->right_x < tapline->left_x + text)
                line_combine(tapline, tap);
            else
                put_section(tap, &(nsection[columnest]));
            tap = tapline;
            continue;
        }
        tap = tap->next;
    }
}

/*
for ( i = 1; i <= columnest; i++){
  printf("section = %d \n", i);
```

【図107】

```
for (tmp = section[i].firstlineblock; tmp != NULL; tmp = tmp->next)
  printf("%d upper_y = %d lower_y = %d ", tmp->index, tmp->upper_y, tmp->lower_y);
printf("\n");
*/

printf("text1 = %f \n", text1);
  alignmap = (text1*2.0);
printf("alignmap = %f \n", alignmap);
  numsection = 0;
    section = NULL;

  free((int *)hist_column);
  for (i = 0 ; i < columnnum; i++)
    {
    if (section[i].firstlineblock==NULL) continue;
    currentsection = new_section(headsection);
    if (section == NULL) section = currentsection;
    currentsection->att = TEXT;
    for (tmp = section[i].firstlineblock; tmp != NULL; ){
      if (currentsection->numlineblocks>0){
        currentsection->firstlineblock = tmp;
        currentsection->currentlineblock = tmp;
        tmp->previous = NULL;
        currentsection->numlineblocks++;
        currentsection->upper_y = tmp->upper_y;
        currentsection->lower_y = tmp->lower_y;
        currentsection->right_x = tmp->right_x;
        currentsection->left_x = tmp->left_x;
        currentsection->att = tmp->att;
        tmp = tmp->next;
        if (tmp != NULL) tmp->previous->next = NULL;
      } else{ if (!((tmp->upper_y-currentsection->lower_y)<sin(tmp->length,
                currentsection->currentlineblock->length)) /* ){*/
                                                                    ||
        (abs(tmp->left_x, currentsection->left_x)>(int)alignmap
                                                                    ||
        (abs(tmp->right_x, currentsection->right_x)>(int)alignmap
                                                                    ||
        (tmp->att!=CAPTION)){
          currentsection = new_section(headsection);
          currentsection->att = TEXT;
        } else{
          currentsection = currentsection->currentlineblock->next;
          currentsection->att = TEXT;
        }
        if ((currentsection->lower_y < tmp->lower_y)
          currentsection->lower_y = tmp->lower_y;
        if (currentsection->right_x < tmp->right_x)
          currentsection->right_x = tmp->right_x;
        if (currentsection->left_x > tmp->left_x)
          currentsection->left_x = tmp->left_x;
        if (cover_range(tmp->upper_y, tmp->lower_y,
          currentsection->currentlineblock->upper_y,
          currentsection->currentlineblock->lower_y)
          linecombine(currentsection->currentlineblock,
              tmp);
          tmp = tmp->next;
        } else {
        currentsection->currentlineblock->next = tmp;
        tmp->previous = currentsection->currentlineblock;
        currentsection->currentlineblock = tmp;
        currentsection->numlineblocks++;
        tmp = tmp->next;
        }
      }
    }
  }
}

for (tmp = newsection[columnnum].firstlineblock; tmp != NULL; ){
  currentsection = new_section(headsection);
  currentsection->att = tmp->att;
  if (section == NULL) section = currentsection;
  if (section == NULL) section = currentsection;
  currentsection->firstlineblock = tmp;
  currentsection->currentlineblock = tmp;
  tmp->previous = NULL;
  currentsection->numlineblocks++;
  currentsection->upper_y = tmp->upper_y;
  currentsection->lower_y = tmp->lower_y;
  currentsection->right_x = tmp->right_x;
  currentsection->left_x = tmp->left_x;
  tmp = tmp->next;
  if (tmp != NULL) tmp->previous->next = NULL;
}
/*
for (currentsection = section; currentsection != NULL;
                                        currentsection = currentsection->next)
  printf(" %1 = %d  %1 = %d %2 = %d y2 = %d att =%x \n", currentsection->left_x,
      currentsection->upper_y, currentsection->right_x,
      currentsection->lower_y, currentsection->att);
*/
  free((struct sectionblock *)section);
  printf("get out of formsection \n");
    return section;
}

/*****************************************************************************/
/*..... Test if the text stick with the horizontal line                      .....*/
/*****************************************************************************/
void str_hold1(first, last, line_y_sort, boundaryline)
  int  first, last;
  struct lineblock **line_y_sort;
  struct lineblock *boundaryline;
{
  int i;
  int distance;

  distance = boundaryline->length-1;
  for (i = first; i <= last; i++){
    if(((line_y_sort[i]->lower_y < boundaryline->lower_y)&&
        (line_y_sort[i]->lower_y > boundaryline->upper_y-distance))(
      if (((line_y_sort[i]->left_x -- boundaryline->left_x - text) &&
          (line_y_sort[i]->right_x <= boundaryline->right_x + text))
        distance = boundaryline->upper_y - line_y_sort[i]->lower_y;
      if (distance < 0)  break;
    }
  }
  if ( distance <= 0) boundaryline->att |= TEXT;
  else {
    if (boundaryline->length >= text)
      boundaryline->att |= (TEXT|RECTEXT);
    else
      boundaryline->att |= PICTURE; /*????? needs more consideration */
  }
}
```

【図108】

```
/*.....remove section from some head section.....*/
/*.....remove nontext section from some head section.....*/
/*.....remove section from some head section.....*/

void rm_ntsection(head, section)
struct sectionblock *head, *section;
{

    if (head->currentnontextsection == section){
        if (section->previous != NULL)
            section->previous->next = NULL;
        else
            head->firstnontextsection = NULL;
        head->currentnontextsection = section->previous;
    } else if (head->firstnontextsection == section){
        head->firstnontextsection = section->next;
        section->next->previous = NULL;
    } else {
        section->next->previous = section->previous;
        section->previous->next = section->next;
    }
}

/*.....remove section from some head section.....*/
/*.....remove section from some head section.....*/
/*.....remove section from some head section.....*/

void rm_section(head, section)
struct sectionblock *head, *section;
{

    if (head->currentsubsection == section){
        if (section->previous != NULL)
            section->previous->next = NULL;
        else
            head->firstsubsection = NULL;
        head->currentsubsection = section->previous;
    } else if (head->firstsubsection == section){
        head->firstsubsection = section->next;
        section->next->previous = NULL;
    } else {
        section->next->previous = section->previous;
        section->previous->next = section->next;
    }
}

/*.....combine two section.....*/
/*.....combine two section.....*/
/*.....combine two section.....*/

void combinesect(first, second, head)
struct sectionblock *first, *second, *head;
{
    printf("combine %d => %d \n", second->index, first->index);
    first->firstlineblock->previous = second->currentlineblock;
    second->currentlineblock->next = first->firstlineblock;
    first->firstlineblock = second->firstlineblock;
    if (second->upper_y < first->upper_y)
        first->upper_y = second->upper_y;
    if (second->lower_y > first->lower_y)
        first->lower_y = second->lower_y;
    if (second->left_x < first->left_x)
        first->left_x = second->left_x;
    if (second->right_x > first->right_x)
        first->right_x = second->right_x;
    first->width = first->right_x - first->left_x + 1;
    first->length = first->lower_y - first->upper_y +1;
    first->numlineblock += second->numlineblock;

    if ((second->attr&TEXT)||(second->att == (TEXT|CAPTION))){
        rm_section(head, second);
    } else {
        rm_ntsection(head, second);
        free((struct sectionblock *)second);
    }
}

/*.....Test if boundary exists between two sections.....*/
/*.....Test if boundary exists between two sections.....*/
/*.....Test if boundary exists between two sections.....*/

int boundaryin(first, topsec, bottomsec, align)
struct sectionblock *first, *topsec, *bottomsec;
int align;
{
    struct sectionblock *tmp;
    float topcover, copycover, botxcover, botycover;
    float coverlap;
    int   x1, y1, x2, y2, x, y, count;
    unsigned char p;

    printf("ge into boundary in\n");
    coverlap = cover_range(bottomsec->upper_y, bottomsec->lower_y,
                           topsec->upper_y, topsec->lower_y);
    coverlap = 0.0;
    if (topsec->attr&CAPTION){
printf("under_caption = %d x1 = %d x2 = %d \n",
        topsec->firstlineblock->under_caption->index,
        topsec->firstlineblock->under_caption->left_x,
        topsec->firstlineblock->under_caption->right_x);
        if ((bottomsec->left_x <
             topsec->firstlineblock->under_caption->left_x-text))||
            (bottomsec->right_x >
             topsec->firstlineblock->under_caption->right_x+text))
            return 1;
    }
```

【図109】

```
if (yoverlap == 1.0){
  if (alignAtRIGHTCORNER){
    x1 = bottomsec->left_x;
    x2 = topsec->left_x-1;
    y1 = topsec->upper_y;
    y2 = bottomsec->upper_y-1;
  } else {
    x1 = topsec->right_x+1;
    x2 = bottomsec->right_x;
    y1 = topsec->upper_y;
    y2 = bottomsec->upper_y-1;
  }
} else if (alignLEFTBOTTOMCORNER){
    x1 = topsec->left_x;
    x2 = bottomsec->left_x-1;
    y1 = bottomsec->lower_y+1;
    y2 = topsec->lower_y;
  } else if (alignRIGHTBOTTOMCORNER){
    x1 = topsec->left_x;
    x2 = topsec->right_x;
    y1 = topsec->lower_y+1;
    y2 = bottomsec->upper_y-1;
  } else {
    x1=x2=topsec->left_x, x2--;
    y1 = y2 = topsec->upper_y;
  }

  p = TEXTPIXEL((0x01);
/* printf("p = %x \n", p); */

  count = 0;
  for (x = x1; x <= x2; x++)
    for (y = y1; y <= y2; y++){
      if (original_p[y][x]p==0x01) {
        count++;
      }
    }
/*
  if (count > (y2-y1+1)) return 1;
*/
  if (count > 0) return 1;
printf("place 1 x1 = %d x2 = %d y1 = %d y2=%d\n", x1, x2, y1, y2);

/*
  if (x2 > x1)
    for (tmp = first->firstnubsection; tmp != NULL; tmp = tmp->next){
      if (tmp==bottomsec||(tmp->topsec)) continue;
      topycover = cover_range(tmp->upper_y, tmp->lower_y, y1, y2);
      topxcover = cover_range(tmp->left_x, tmp->right_x, x1, x2);
      if (((topxcover == 0.0)||(topycover == 0.0)))
        else if ((topxcover == 1.0)&&(topycover == 1.0))) continue;
          return 1;
      }
*/
  if (yoverlap == 1.0){
    if (alignLEFTBOTTOMCORNER){
      x1 = bottomsec->left_x;
      x2 = topsec->left_x-1;
      y1 = bottomsec->lower_y+1;
      y2 = topsec->lower_y;
    } else {
      x1 = topsec->right_x+1;
      x2 = bottomsec->right_x;
      y1 = bottomsec->lower_y+1;
      y2 = topsec->upper_y;
    }
  } else if (alignRIGHTTOPCORNER){
      x1 = topsec->left_x;
      x2 = bottomsec->left_x-1;
      y1 = topsec->upper_y;
      y2 = bottomsec->upper_y-1;
    } else {
      x1=x2=topsec->left_x, x2--;
      y1 = y2 = topsec->upper_y;
    }

  count = 0;

  for (x = x1; x <= x2; x++)
    for (y = y1; y <= y2; y++){
      if ((original_p[y][x]p==0x01) {
        count++;
      }
    }
/*
  if (count > (y2-y1+1)) return 1;
*/
  if (count > 0) return 1;

printf("place 2 x1 = %d x2 = %d y1 = %d y2=%d\n", x1, x2, y1, y2);

/*
    if (x2 > x1)
    for (tmp = first->firstnubsection; tmp != NULL; tmp = tmp->next){
      if (tmp==bottomsec||(tmp->topsec)) continue;
      topycover = cover_range(tmp->upper_y, tmp->lower_y, y1, y2);
      topxcover = cover_range(tmp->left_x, tmp->right_x, x1, x2);
      if (((topxcover == 0.0)||(topycover == 0.0)))
        else if ((topxcover == 1.0)&&(topycover == 1.0))) continue;
          return 1;
      }
*/
  if (yoverlap == 1.0){
    if (alignLEFTBOTTOMCORNER){
      x1 = bottomsec->left_x;
      x2 = topsec->left_x-1;
      y1 = bottomsec->lower_y+1;
      y2 = topsec->lower_y;
    } else {
      x1 = topsec->right_x+1;
      x2 = bottomsec->right_x;
      y1 = bottomsec->lower_y+1;
```

```
      y2 = topsec->lower_y;
    }
  } else if (alignRIGHTTOPCORNER){
      x1 = bottomsec->left_x;
      x2 = topsec->left_x-1;
      y1 = topsec->upper_y;
      y2 = bottomsec->upper_y-1;
    } else if (alignRIGHTBOTTOMCORNER){
      x1 = topsec->right_x+1;
      x2 = bottomsec->right_x;
      y1 = topsec->upper_y;
      y2 = bottomsec->upper_y-1;
    } else { x1=x2=topsec->right_x, x2--;
      y1 = y2 = topsec->upper_y; }

  count = 0;

  for (x = x1; x <= x2; x++)
    for (y = y1; y <= y2; y++){
      if ((original_p[y][x]p==0x01) {
        count++;
      }
    }
/*
  if (count > (y2-y1+1)) return 1;
*/
  if (count > 0) return 1;

printf("place 3 x1 = %d x2 = %d y1 = %d y2=%d\n", x1, x2, y1, y2);

/*
    if (x2 > x1)
    for (tmp = first->firstnubsection; tmp != NULL; tmp = tmp->next){
      if (tmp==bottomsec||(tmp->topsec)) continue;
      topycover = cover_range(tmp->upper_y, tmp->lower_y, y1, y2);
      topxcover = cover_range(tmp->left_x, tmp->right_x, x1, x2);
      if (((topxcover == 0.0)||(topycover == 0.0)))
        else if ((topxcover == 1.0)&&(topycover == 1.0))) continue;
          return 1;
      }
*/
  if (yoverlap == 1.0){
    if (alignLEFTBOTTOMCORNER){
      x1 = bottomsec->left_x;
      x2 = topsec->left_x-1;
      y1 = bottomsec->lower_y+1;
```

```
      y2 = topsec->lower_y;
    }
  } else if (alignRIGHTBOTTOMCORNER){
      x1 = topsec->right_x+1;
      x2 = bottomsec->right_x;
      y1 = topsec->upper_y;
    } else { x1=x2=topsec->right_x, x2--;
      y1 = y2 = topsec->upper_y; }

  count = 0;

  for (x = x1; x <= x2; x++)
    for (y = y1; y <= y2; y++){
      if ((original_p[y][x]p==0x01) {
        count++;
      }
    }
/*
  if (count > (y2-y1+1)) return 1;
*/
  if (count > 0) return 1;

printf("place 4 x1 = %d x2 = %d y1 = %d y2=%d\n", x1, x2, y1, y2);

/*
    if (x2 > x1)
    for (tmp = first->firstnubsection; tmp != NULL; tmp = tmp->next){
      if (tmp==bottomsec||(tmp->topsec)) continue;
      topycover = cover_range(tmp->upper_y, tmp->lower_y, y1, y2);
      topxcover = cover_range(tmp->left_x, tmp->right_x, x1, x2);
      if (((topxcover == 0.0)||(topycover == 0.0)))
        else if ((topxcover == 1.0)&&(topycover == 1.0))) continue;
          return 1;
      }
*/
  if (yoverlap == 1.0){
    if (alignLEFTBOTTOMCORNER){
      x1 = bottomsec->left_x;
      x2 = topsec->left_x-1;
      y1 = bottomsec->lower_y+1;
```

```
      y2 = topsec->lower_y;
    }
  } else if (alignRIGHTBOTTOMCORNER){
      x1 = topsec->left_x;
      x2 = bottomsec->left_x-1;
      y1 = topsec->upper_y;
      y2 = bottomsec->upper_y-1;
    } else { x1=x2=topsec->left_x, x2--;
      y1 = y2 = topsec->upper_y; }

  count = 0;

  for (x = x1; x <= x2; x++)
    for (y = y1; y <= y2; y++){
      if ((original_p[y][x]p==0x01) {
        count++;
      }
    }
/*
  if (count > (y2-y1+1)) return 1;
*/
  if (count > 0) return 1;

printf("place 5 x1 = %d x2 = %d y1 = %d y2=%d\n", x1, x2, y1, y2);

/*
    if (x2 > x1)
    for (tmp = first->firstnubsection; tmp != NULL; tmp = tmp->next){
      if (tmp==bottomsec||(tmp->topsec)) continue;
      topycover = cover_range(tmp->upper_y, tmp->lower_y, y1, y2);
      topxcover = cover_range(tmp->left_x, tmp->right_x, x1, x2);
      if (((topxcover == 0.0)||(topycover == 0.0)))
        else if ((topxcover == 1.0)&&(topycover == 1.0))) continue;
          return 1;
      }
*/
  if ((tmp->lower_y >= topsec->upper_y)&&
      (tmp->upper_y <= bottomsec->lower_y)&&
      (tmp->left_x <= max(topsec->right_x, bottomsec->right_x))&&
```

【図１１０】

```
    (tmp->right_x >= min(topsec->left_x, bottomsec->left_x))
    return 0;

  if (if_overlap(tmp->upper_y, tmp->lower_y,
        min(topsec->upper_y, bottomsec->upper_y),
        max(topsec->lower_y, bottomsec->lower_y))){
    if ((if_overlap(topsec->left_x, topsec->right_x,
        tmp->left_x, tmp->right_x)
      && (if_overlap(topsec->left_x, bottomsec->left_x,
        min(topsec->right_x, bottomsec->right_x)))
    ){
      return 1;
    }
  }
  return 0;
}

/*.....*/
/*..... Test if some section fully inside the other section .....*/
/*.....*/

int full_inside(first, second)
struct sectionblock *first, *second;
{
  if ((first->left_x >= second->left_x)&&(first->right_x <= second->right_x)&&
      (first->upper_y >= second->upper_y)&&(first->lower_y <= second->lower_y))
    return 1;
  else
    return 0;
}

/*.....*/
/*..... Test if two section are neighbors (belong to same column)  .....*/
/*.....*/

int save_section(sort_y, first, sec, num)
struct sectionblock **sort_y;
int   first, sec, num;
{
  int   x1, y1, x2, y2, i, j, count;
  int   in_segment, in_column, in_segment, in_column;
  struct sectionblock *tmp1, *tmp2;
  int   area1_y1, area1_y2, area2_y1, area2_y2;
  int   area1_x1, area1_x2, area2_x1, area2_x2;
  char  p;

  printf("in save_section index 1= %d index 2 = %d \n", sort_y[first]->index,
      sort_y[sec]->index);

  if (sort_y[first]->left_x < sort_y[sec]->left_x){
    tmp1 = sort_y[first];
    tmp2 = sort_y[sec];
  } else {
    tmp1 = sort_y[sec];
    tmp2 = sort_y[first];
  }

  if ((if_overlap(tmp1->upper_y, tmp1->lower_y,
        tmp2->upper_y, tmp2->lower_y) == 0)) return 0;

  if ((tmp1->attrCAPTION)&&(tmp2->attrCAPTION)){
    if (tmp1->firstlineblock->node_caption !=
        tmp2->firstlineblock->node_caption){
      return 0;
    }
  } else if ((tmp1->attrCAPTION)||(tmp2->attrCAPTION)){
    return 0;
  }

  in_segment = tmp1->boundary_index;
  in_column = 0;
  in_segment = tmp2->boundary_index;
  in_column = 0;

  for (j = 1, i = in_segment; j <= segment[i].column_no; j++){

/*  printf("j = %d, x1 = %d x2 = %d, column_x1 = %d column_x2 = %d \n", j,
    tmp1->left_x, tmp2->right_x, segment[i].begin_column[j],segment[i].end_column[j]);
*/
    if (cover_range(tmp1->left_x, tmp1->right_x,
        segment[i].begin_column[j], segment[i].end_column[j])==1.0){
      in_column = j;
      break;
    }
  }

/*  printf("seg1 = %d coll = %d seg2 = %d coll = %d\n", in_segment, in_column,
      in_segment, in_2column);
*/
  for (j = 1, i = in_segment; j <= segment[i].column_no; j++){
    if ((cover_range(tmp2->left_x, tmp2->right_x,
        segment[i].begin_column[j], segment[i].end_column[j])==1.0 )&&
        (in_column != j)){
      in_2column = j;
      break;
    }
  }

/*  printf("seg1 = %d coll = %d seg2 = %d col2 = %d\n", in_segment, in_column,
      in_segment, in_2column);
*/
  if ((in_segment >= 0)&&(in_segment >= 0)){
    if ((in_segment != in_segment))
    {
/*    printf("different segment\n");
      if ((in_column>0)&&(in_2column>0)){
        if ((cover_range(tmp2->left_x, tmp2->right_x,
            segment[in_segment].begin_column[in_column],
            segment[in_segment].end_column[in_column])<1.0 )&&
            (cover_range(segment[in_segment].begin_column[in_2column],
            segment[in_segment].end_column[in_2column],
            tmp1->left_x, tmp1->right_x)<1.0)){
          return 0;
        } else
          return 0;
      } else {
        if ((tmp2->left_x<tmp1->right_x)&&(int)(text1) ||
            (tmp1->width > (int)(text1*3.0))||
            (tmp2->width > (int)(text1*3.0))){
          return 0;
        }
        printf("same segment\n");
      }
    } else
      return 0;
  }
/* printf("same segment, same column\n"); */
  x1 = tmp1->right_x+1;
  x2 = tmp2->left_x-1;
  if (tmp1->upper_y >= tmp2->upper_y){
```

【図111】

```
y1 = tmp2->upper_y;
area1_x1 = tmp1->left_x;
area1_x2 = tmp2->left_x-1;
area1_y1 = y1;
area1_y2 = tmp1->upper_y-1;
} else {
        y1 = tmp1->upper_y;
        area1_x1 = tmp1->right_x+1;
        area1_x2 = tmp2->right_x;
        area1_y1 = y1;
        area1_y2 = tmp1->upper_y-1;
    }

    if (tmp1->lower_y == tmp2->lower_y){
        y1 = tmp1->lower_y;
        area2_x1 = tmp1->right_x+1;
        area2_x2 = tmp2->right_x;
        area2_y1 = tmp1->lower_y;
        area2_y2 = tmp1->lower_y;
    } else {
        y1 = tmp2->lower_y;
        area2_x1 = tmp2->left_x;
        area2_x2 = tmp2->left_x+1;
        area2_y1 = tmp2->lower_y+1;
        area2_y2 = tmp2->lower_y;
    }

    p = TEXTFIELD(0x01);

    if (in_segment == in_segment(&(in_column [* in_column))
    return 0;
    if (!in_segment || !(in_column != in_column)){
    if (((x2-x1)>(int)(text2/2)&&((tmp1->width > (int)(2.0*text))&&
                (tmp1->width > (int)(2.0*text))))) return 0;
    }
    for ( i = x1; i <= x2; i++)
        for (j = y1; j <= y2; j++)
            if ((original_p[i][j][i]p) return 0;

/* printf("no text between\n"); */
    if (((x2-x1) < 6)||((in_segment == in_segment)&&(in_column == in_column)
        && ((tmp2->left_x-tmp1->right_x)<(int)(text*1.2)))){

/* printf("narrow gap \n"); */ area2_x1; i++){
        for (i = area1_x1; i <= area1_x2; i++){
            for (j = area1_y1; j <= area1_y2; j++){
                if ((original_p[i][j][i]p)) return 0;
            }
        }
/* printf("place 1\n"); */
        for (i = area2_x1; i <= area2_x2; i++){
            for (j = area2_y1; j <= area2_y2; j++){
                if ((original_p[i][j][i]p)) return 0;
            }
        }
        return 1;
    } else return 0;
}

/*****************************************************************/
/*****  test if two text block overlap with the two combined block  *****/
/*****************************************************************/

int text_intersect(header, first, second)
struct sectionblock *header;
struct sectionblock *first, *second;
{
    struct sectionblock *sectionblock;
    int   x1, y1, x2, y2;
    float xcover, ycover, xcover2;

/*
printf("get into text inter first = %d second = %d \n",
                    first->index, second->index);
*/
    if ((first->upper_y > second->upper_y){
        tmpsec = first;
        first = second;
        second = tmpsec;
    }
    x1 = min(first->left_x, second->left_x);
    x2 = max(first->right_x, second->right_x);
    y1 = min(first->upper_y, second->upper_y);
    y2 = max(first->lower_y, second->lower_y);

    for (tmpsec = header->firstsubsection; tmpsec != NULL; tmpsec = tmpsec->next){
/*
printf("%d x1 = %d y1 = %d ", tmpsec->index, tmpsec->left_x, tmpsec->upper_y);
*/
        if ((tmpsec->upper_y < first->lower_y)&&
                (tmpsec->lower_y > second->upper_y)) return 1;
        if ((tmpsec->upper_y > first->lower_y){{
            if ((tmpsec->lower_y <= second->lower_y-(int)text)&&
                (xcover2=-1.0)){
                if (tmpsec->lower_y > y2)
                    y2 = tmpsec->lower_y;
                if (tmpsec->left_x < x1) {
                    x1 = tmpsec->left_x;
                }
                if (tmpsec->right_x > x2) {
                    x2 = tmpsec->right_x;
                }
                continue;
            } else return 1;
        } else if ((tmpsec->lower_y <= second->upper_y){
/* printf("xcover2 = %d \n", xcover2; first->upper_y-(int)text)); */ ~
                if ((tmpsec->upper_y > first->upper_y-(int)text)&&
                    (xcover2 == -1.0)) {
                    if (tmpsec->upper_y < y1)
                        y1 = tmpsec->upper_y;
                    if (tmpsec->left_x < x1) {
                        x1 = tmpsec->left_x;
                    }
                    if (tmpsec->right_x > x2) {
                        x2 = tmpsec->right_x;
                    }
                    continue;
                } else return 1;
            }
        }
    }
    if (x1 < min(first->left_x, second->left_x))
    second->left_x = x1;
    if (x2 > max(first->right_x, second->right_x))
    second->right_x = x2;
```

【図112】

```
    if (y1 < min(first->upper_y, second->upper_y))
        first->upper_y = y1;
    if (y2 > max(first->lower_y, second->lower_y))
        second->lower_y = y2;
    return 0;
}

/*
 *.....................................................................
 *                                                                     *
 *.....Test if the text block with the horizontal line                *
 *                                                                     *
 *.....................................................................
 */

void section_combine(first, boundary)
struct sectionblock *first;
struct linelock     *boundary;
{
    int   i, j, b, sectionsep, align, if_combine, num;
    float alignsep;
    struct sectionblock **sort_y_section;

printf("get into section combine \n");

    sectionsep = (int)((float)text*1.5);
    alignsep = text*2.0;
    sort_y_section = sort_section(first, &num);

printf("after sort_y section. num = %d \n", num);

for (i = 0; i < num; i++)
printf("i = %d index = %d \n", i, sort_y_section[i]->index);
    if (sort_y_section[i]->att == (TEXTTITLE)) continue;
    if (sort_y_section[i]->att&CAPTION) continue;
    for (j = i-1; j >= 0; j--){
        if ((sort_y_section[j] == NULL)||
            (sort_y_section[j]->att == (TEXTTITLE))) continue;
        if_combine = 0;
        if (((float)(sort_y_section[i]->upper_y - sort_y_section[j]->lower_y) <
            (float)min(sort_y_section[i]->height, sort_y_section[j]->height)*0.5)){
            if_combine = 1;
        }

        if ((if_overlap(sort_y_section[i]->left_x, sort_y_section[i]->right_x,
            sort_y_section[j]->left_x, sort_y_section[j]->right_x))){
            align = 0;
            if (sort_y_section[i]->left_x < sort_y_section[j]->left_x)
                align |= LEFTTOLONGER;
            else if (sort_y_section[i]->left_x > sort_y_section[j]->left_x)
                align |= LEFTTOFONGER;
            if ((float)abs(sort_y_section[i]->left_x -
                           sort_y_section[j]->left_x) <= alignsep)
                align |= LEFTALIGN;
            if (sort_y_section[i]->right_x > sort_y_section[j]->right_x)
                align |= RIGHTTOLONGER;
            else if (sort_y_section[i]->right_x < sort_y_section[j]->right_x)
                align |= RIGHTTOFONGER;
            if ((float)abs(sort_y_section[i]->right_x -
                           sort_y_section[j]->right_x) <= alignsep)
                align |= RIGHTALIGN;
            if (sort_y_section[i]->right_x > sort_y_section[j]->right_x)

/*
 *.....Test if the text block with the horizontal line                *
 */

            if ((sort_y_section[i]->upper_y <= sort_y_section[j]->lower_y*  /*
                (int)text*1.5)){

                min(sort_y_section[i]->firstlineblock->length,
                sort_y_section[j]->currentlineblock->length)){

        if (((if_overlap(sort_y_section[i]->left_x, sort_y_section[i]->right_x,
            sort_y_section[j]->left_x, sort_y_section[j]->right_x))){
            align = 0;
            if (sort_y_section[i]->left_x < sort_y_section[j]->left_x)
                align |= LEFTTOLONGER;
            else if (sort_y_section[i]->left_x > sort_y_section[j]->left_x)
                align |= LEFTTOFONGER;
            if ((float)abs(sort_y_section[i]->left_x -
                           sort_y_section[j]->left_x) <= alignsep)
                align |= LEFTALIGN;
            if (sort_y_section[i]->right_x > sort_y_section[j]->right_x)
                align |= RIGHTTOLONGER;
            else if (sort_y_section[i]->right_x < sort_y_section[j]->right_x)
                align |= RIGHTTOFONGER;
            if ((float)abs(sort_y_section[i]->right_x -
                           sort_y_section[j]->right_x) <= alignsep)
                align |= RIGHTALIGN;

            if ((align&LEFTALIGN)&&(align&LEFTTOLONGER))||
                ((align&RIGHTALIGN)&&(align&RIGHTTOLONGER)))||
                ((align&LEFTTOLONGER)&&(align&RIGHTTOFONGER)))
                if_combine = 2;

printf("i = %d j = %d align= %xin", i, j, align);
            } else if ((cover_range(sort_y_section[i]->upper_y,
                                    sort_y_section[j]->upper_y,
                                    sort_y_section[j]->lower_y)>0.3)
                                    ||(cover_range(sort_y_section[j]->upper_y,
                                    sort_y_section[i]->lower_y, sort_y_section[i]->upper_y,
                                    sort_y_section[i]->lower_y)>0.5)){

printf("j = %d i = %d \n", j, i);
                if (same_section(sort_y_section, j, i, num))
                    if_combine = 1;

printf("after same_section\n");
                }
            }

if (((if_combine == 1)||((if_combine == 2)) {

    if (((cover_range(sort_y_section[i]->lower_y,
        sort_y_section[j]->lower_y,
        sort_y_section[j]->lower_y)
        /
        (sort_y_section[j]->lower_y - sort_y_section[j]->upper_y)) > 0.5)){

        continue;
    }

    if (sort_y_section[j]->lower_y < sort_y_section[i]->upper_y)
    if (!sec_inerase(first, sort_y_section[j], sort_y_section[i]))
        continue;

sort_y_section[i]->att |= sort_y_section[j]->att;
combinesec(sort_y_section[i], sort_y_section[j], first);
sort_y_section[j] = NULL;
            }
        }
    }

    if ((align&RIGHTALIGN) == RIGHTALIGN){
        switch(align&LEFTALIGN) {
        case LEFTALIGN:
            if_combine = 1;
            break;
        case LEFTTOFONGER:
            if_combine = 0;
            break;
        case LEFTTOLONGER:
            if (line_no(sort_y_section[i]) == 1)
                if_combine = 1;
            else
                if_combine = 0;
            break;
        }
```

【図113】

【図114】

```
/*.....test if located within the hpicture                    .....*/
/*.....                                                       .....*/
/*.....................................................................*/

int if_inhpicture(tmp1, tmp2)
struct linkblock *tmp1;
struct sectionblock *tmp2;
{
    int x, y, count;
    unsigned char p;

    p = TEXTURE[YSIZE];
    for (y = tmp1->upper_y; y <= tmp1->lower_y; y++){
        count = 0;
        for (x = tmp1->left_x-1; x >= tmp2->left_x; x--)
            if ((original_p[y][x]&p)==0x01) count++;
/* printf("count 1 = %d \n", count); */
        if (count == ((int)(HPOINT/SCALE_RATIO)-2)) continue;
        count = 0;
        for (x = tmp1->right_x+1; x <= tmp2->right_x; x++)
            if ((original_p[y][x]&p)==0x01) count++;
/* printf("count 2 = %d \n", count); */
        if (count > ((int)(HPOINT/SCALE_RATIO)-2)) return 1;
    }

    for (x = tmp1->left_x; x <= tmp1->right_x; x++){
        count = 0; tmp1->upper_y-1; y >= tmp2->upper_y; y--)
        for ( y = ((original_p[y][x]&p)==0x01) count++;
        if (count == ((int)(HPOINT/SCALE_RATIO)-2)) continue;
        count = 0;
        for ( y = tmp1->lower_y+1; y <= tmp2->lower_y; y++
            if ((original_p[y][x]&p)==0x01) count++;
        if (count >= ((int)(HPOINT/SCALE_RATIO)-2)) return 1;
    }
    return 0;
}

/*.....test if text line has some neighbor in same section    .....*/
/*.....                                                       .....*/
/*.....................................................................*/

int no_line_neighbor(tmp1, tmp2)
struct sectionblock *tmp2;
struct linkblock    *tmp1;
{
    struct linkblock *tmp_line;

printf("tmp1 = %d x1 = %d y1 = %d x2 = %d y2 = %d \n", tmp1->index,
        tmp1->left_x, tmp1->right_x,tmp1->lower_y);
    for (tmp_line = tmp2->firstlineblock; tmp_line != NULL;
                tmp_line = tmp_line->next){

    if (tmp_line == tmp1) continue;
printf("tmp_line = %d x1 = %d y1 = %d x2 = %d y2 = %d \n", tmp_line->index,
        tmp_line->left_x, tmp_line->upper_y, tmp_line->right_x,tmp_line->lower_y);

    if (((((overlap(tmp_line->upper_y,(int)(text2*1.5),
            tmp_line->lower_y,(int)(text1*1.5),
        ((((overlap(tmp_line->left_x-tmp_line->length,
            tmp_line->right_x+tmp_line->length,
            tmp1->left_x, tmp1->right_x))){
        if ((tmp_line->length >= (int)(text1*2))&&
                (tmp_line->width >= (int)(text1*2))) return 0;
        }
    }
    return 1;
}

/*.....................................................................*/
/*.....test if inside another text block or nontext block      .....*/
/*.....                                                       .....*/
/*.....................................................................*/

void inside_text(firstsection)
struct sectionblock *firstsection;
{

    struct sectionblock *tmp2, *tmpn2, *tmpn;
    struct linkblock    *tmp1, *tmpl;
    int rearrange;
    int within_pic;
    float xcover, ycover, xcover2;
    float cover1_x, cover1_y, cover2_x, cover2_y;
    int   v, count;
/*
printf("get into inside text\n");
*/
    for (tmp = firstsection->firstnontextsection; tmp != NULL; ){
                                        tmpn = tmp->next;
        for (tmp2 = firstsection->firstnontextsection; tmp2 != NULL; ){
                                        tmpn2 = tmp2->next;
/*
printf("tmp = %d \n", tmp->index);
*/
            if (tmp2 == tmp) break;
            tmpl = tmp2->firstlineblock->first_block->next;
            no_set = 0;
printf("tmp2 = %d att = %x cover = %d \n", tmp2->index, tmp2->att,
                                        tmp2->firstlineblock->first_block->cover);
            if ((((tmp2->att&(PICTURE|TABLE|FRAME))!=0)||((tmp2->att&HPICTURE)&&
                                        (tmp2->firstlineblock->first_block->cover ==0)))
                continue;
            if (tmp2->upper_y > tmp->lower_y) break;

            xcover = cover_range(tmp->left_x, tmp->right_x, tmp2->left_x,
                                        tmp2->right_x);
            xcover2 = cover_range(tmp->left_x, tmp->right_x,
                                        tmp2->left_x-tmp2->length/2,
                                        tmp2->right_x+tmp2->length/2);
            ycover = cover_range(tmp->upper_y, tmp->lower_y, tmp2->upper_y,
                                        tmp2->lower_y);

printf("tmp2 = %d xcover = %f ycover = %f \n", tmp2->index, xcover, ycover);

            if (((xcover > 0.0)||(xcover2 > 0.0))&&(ycover > 0.0)){
                tmp1 = tmp->next;
                if ((tmp2->att&TABLE)&&(ycover >= 0.5)&&(xcover2 > 0.0)) {

                if ((tmp2->att&(TABLE|FRAME))&&(ycover >= 0.5)&&(xcover2 > 0.0166
                        (tmp->length<tmp2->length)) {

                    combinesec(tmp2, tmp, firstsection);
                    no_set = 1;
                    tmp = tmp2;
                    break;
                }
                else
                if ((((tmp2->att&(HSOLID|DOWN))==(HSOLID|DOWN)))||
                    (((tmp2->att&(HSOLID|UP))==(HSOLID|UP)))&&(xcover > 0.0166
                    (tmp2->length >= (int)(text1/2)&&
                    (cover_range(tmp2->upper_y, tmp2->lower_y, tmp->upper_y,
                                        tmp->lower_y)>0.5)){
                    tmp2->att |= TEXT;
                    no_set = 1;
                    tmp = tmp2;
                }
            }
        }
        tmp = tmpn;
    }
    return 1;
}
```

【図115】

```
            if !(tmpnt->att&VSOLID)||(tmpnt->att&HSOLID)){
               if ((v == HPICTURE)&&(cover1_x == 1.0)&&(cover1_y == 1.0)){
                  combine = 3;
                  tmpnt->att = HPICTURE;
               } else{
                        if ((v == PICTURE)&&(cover1_x == 1.0)&&(cover1_y == 1.0))
                           tmpnt->att = v;
                        continue;
                     }
               }
               if (combine != 3){
/* printf("combine %d and %d \n", tmpnt->index, tmpnt2->index); */
                  if (combine == 1)
                     bline_combine(tmpnt->firstlineblock, tmpnt2->firstlineblock);
                  else if (combine == 2)
                     fline_combine(tmpnt->firstlineblock, tmpnt2->firstlineblock);
                  tmps = tmpnt2->previous;
                  rm_ntsection(firstsection, tmpnt2);
                  tmpnt2 = tmps;
                  tmpnt->left_x = tmpnt->firstlineblock->left_x;
                  tmpnt->right_x = tmpnt->firstlineblock->right_x;
                  tmpnt->upper_y = tmpnt->firstlineblock->upper_y;
                  tmpnt->lower_y = tmpnt->firstlineblock->lower_y;
                  tmpnt->length = tmpnt->lower_y-tmpnt->upper_y+1;
                  tmpnt->width = tmpnt->right_x-tmpnt->left_x+1;
               }
            }
         }
      }
}

/*************************************************************************/
/******** Section formation for each head line block             ********/
/*************************************************************************/

void section_block(firstsection,  firstline,  column)
struct sectionblock *firstsection;
struct lineblock    *firstline;
int    column;
{
     int    i, j, numline;
     int    firstln, lastln, nextfirst;
     int    num_bound, numsection, numnontextsection;
     struct lineblock **line_y_sort, **boundary;
     struct lineblock *tmpline;
     struct sectionblock *tmpsection, *subcurrentsec;


 printf("get into section block index = %d x=%d y = %d \n", firstline->index,
                    firstline->left_x, firstline->upper_y);
     min_y = firstline->upper_y;
     max_y = firstline->lower_y;
     textl = (firstline->first_block->avg_height == 0.0)?
                    text_length:firstline->first_block->avg_height;
/*     textl = avg_line_length(firstline); */
printf("textl = %f \n", textl);
     line_y_sort = sort_y(firstline, &numline);
printf("after line_y_sort numline = %d \n", numline);
     title_scan(line_y_sort, firstline, &numline);
printf("after title_scan numline = %d \n", numline);
     boundary = boundary_form(line_y_sort, &num_bound, &numline);
printf("after boundary_form numline = %d num_bound = %d \n", numline, num_bound);

     segment = new_segment(0, num_bound);
     current_seg = 0;
     firstln = lastln = nextfirst  = 0;
     for (numsection = 0, numnontextsection = 0, i = 0;
                                                  (i < num_bound); i++){
printf("boundary = %d \n", boundary[i]->index);
        current_seg = i;
        firstln = lastln = nextfirst;
        for (j = firstln ; (j < numline)&&
             (line_y_sort[j]->lower_y <= boundary[i]->upper_y); lastln = j, j++);
           nextfirst = j;
/*
           if ((boundary[i]->att&HSOLID)&&(boundary[i]->att&DOWN))
              att_hsolid(firstln, lastln, line_y_sort, boundary[i]);
*/

           if (lastln != nextfirst){
printf("firstln = %d lastln = %d \n", firstln, lastln);
              subcurrentsec = form_section(firstsection, firstln, lastln,
                                                   line_y_sort, column, 0);
              for (tmpsection = subcurrentsec; tmpsection != NULL;
                                                   tmpsection = tmpsection->next){
                 tmpsection->index = ++numsection;
                 tmpsection->width = tmpsection->right_x - tmpsection->left_x+1;
                 tmpsection->length = tmpsection->lower_y-tmpsection->upper_y+1;
                 tmpsection->boundary_index = i;
              }
           }
           if (boundary[i]->att == (TEXT|TITLE)){
              subcurrentsec = new_ntsection(firstsection);
              subcurrentsec->index = ++numnontextsection;
           } else {
                    subcurrentsec = new_ntsection(firstsection);
                    subcurrentsec->index = ++numnontextsection;
                  }
           subcurrentsec->firstlineblock =
                          subcurrentsec->currentlineblock = boundary[i];
           subcurrentsec->upper_y = boundary[i]->upper_y;
           subcurrentsec->lower_y = boundary[i]->lower_y;
           subcurrentsec->left_x = boundary[i]->left_x;
           subcurrentsec->right_x = boundary[i]->right_x;
           subcurrentsec->width = boundary[i]->width;
           subcurrentsec->length = boundary[i]->length;
           subcurrentsec->att = boundary[i]->att;
           subcurrentsec->att &= (~UNKNOWN);
           boundary[i]->previous = NULL;
           boundary[i]->next = NULL;
        }

printf("nextfirst = %d numline = %d\n", nextfirst, numline);
     if (nextfirst < numline) {
        current_seg = num_bound;
        firstln = nextfirst;
        lastln = numline-1;
        subcurrentsec = form_section(firstsection, firstln, lastln,
                                          line_y_sort, column, 1);
        for (tmpsection = subcurrentsec; tmpsection != NULL;
                                          tmpsection = tmpsection->next){
           tmpsection->width = tmpsection->right_x - tmpsection->left_x+1;
           tmpsection->length = tmpsection->lower_y-tmpsection->upper_y+1;
           tmpsection->index = ++numsection;
           tmpsection->boundary_index = current_seg;
        }
     }

     if (firstsection->firstsubsection != NULL){
printf("before attribute_text \n");
     attribute_text(firstsection);
printf("after attribute_text \n");
     for ( i = 0; i <= num_bound; i++){
        printf("segment  = %d no_seg = %d ", i, segment[i].column_no);
```

【図117】

```
            for ( j = 1; j <= segment[i].column_no; j++)
                printf("bg = %d and = %d ", segment[i].begin_column[j],
                                            segment[i].end_column[j]);
            printf("\n");
        }

/*      section_separate(firstsection); */

printf("before section combine\n");
/*
        section_combine(firstsection, boundary);
*/

        section_combine(firstsection, boundary);


        inside_text(firstsection);


        inside_text(firstsection);


    }
    free((struct segment_gap *)segment);
    free((struct lineblock **)line_y_sort);
    free((struct lineblock **)boundary);

/*
        section_combine(firstsection, boundary);
*/
/*      inner_section_combine(firstsection); */
}

/**************************************************************************/
/******** Table test                                              ********/
/**************************************************************************/
void  move_line_to_section(firstsection, firstline)
struct sectionblock *firstsection;
struct lineblock    *firstline;
{
      struct lineblock *tmpline, *tmpl1;
      struct sectionblock *currentsection;
      int i;

printf("get into move \n");
      for (i = 0, tmpline = firstline->firstsubline; tmpline != NULL; )
          {
              currentsection = new_section(firstsection);
              currentsection->firstlineblock = currentsection->currentlineblock
                                             = tmpline;
              currentsection->left_x = tmpline->left_x;
              currentsection->right_x = tmpline->right_x;
              currentsection->upper_y = tmpline->upper_y;
              currentsection->lower_y = tmpline->lower_y;
              currentsection->width = tmpline->width;
              currentsection->length = tmpline->length;
              currentsection->index = ++i;
              currentsection->att = TEXT;
              firstsection->numlineblock++;
              tmpl1 = tmpline->next;
              tmpline->next = NULL;
              tmpline->previous = NULL;
              tmpline = tmpl1;
          }

printf("before end\n");
      for (i = 0, tmpline = firstline->firstnontextline; tmpline != NULL; )
          {
              currentsection = new_ntsection(firstsection);
              currentsection->firstlineblock = currentsection->currentlineblock
                                             = tmpline;
              currentsection->left_x = tmpline->left_x;
              currentsection->right_x = tmpline->right_x;
              currentsection->upper_y = tmpline->upper_y;
              currentsection->lower_y = tmpline->lower_y;
              currentsection->width = tmpline->width;
              currentsection->length = tmpline->length;
              currentsection->index = ++i;
              currentsection->att = tmpline->att;
              tmpl1 = tmpline->next;
              tmpline->next = NULL;
              tmpline->previous = NULL;
              tmpline = tmpl1;
          }
/*
printf("in move \n");
for (currentsection=firstsection->firstsubsection; currentsection != NULL;
                                       currentsection = currentsection->next)
printf("index = %d x = %d y = %d \n", currentsection->index, currentsection->left_x,
                                       currentsection->upper_y);
for (currentsection=firstsection->firstnontextsection; currentsection != NULL;
                                       currentsection = currentsection->next)
printf("index = %d x = %d y = %d \n", currentsection->index, currentsection->left_x,
                                       currentsection->upper_y);
*/
printf("get out of move \n");
}

/**************************************************************************/
/******** Test if the text stick with the horizontal line         ********/
/**************************************************************************/

void ini_section_block(firstsection, firstline, p, column)
struct sectionblock *firstsection;
struct lineblock *firstline;
unsigned char **p;
int     column;
{
      struct sectionblock *tmpsection;
      int     num;
      struct lineblock *tmpline;


      original_p = p;
      firstsection->firstlineblock = firstsection->currentlineblock = firstline;
/*printf("firstline index = %d text = %d nontext = %d group = %d \n",
      firstline->index,    firstline->firstsubline, firstline->firstnontextline,
      firstline->group); */
      if ((firstline->firstsubline != NULL)||(firstline->firstnontextline != NULL)){
          if ((firstline->att&TABLE)||((firstline->att&PICTURE)&&
                         (firstline->group >= 2)&&(!(firstline->att&FRAME))))
             move_line_to_section(firstsection, firstline);
          else
             section_block(firstsection, firstline, column);

          firstline->firstsubline = NULL;
          firstline->firstnontextline = NULL;

      }

      for (tmpsection = firstsection->firstnontextsection; tmpsection != NULL;
```

【図119】

```
/*****************************************************************/
/*                                                               */
/*                File: table.c                                  */
/*                Author: Shin-Yuan Wang                         */
/*                Date : 3-6-92                                  */
/*  Copyright 1992 Canon Information Systems                     */
/*                                                               */
/*****************************************************************/

#include <stdio.h>
#include <malloc.h>
#include "table.h"

extern void exit();

/*****************************************************************/
/*                                                               */
/*  allocate new white_for_table structure                       */
/*                                                               */
/*****************************************************************/
struct white_for_table *new_wtable(wblock)
struct whiteblock *wblock;
{
    struct white_for_table *v;

    v = (struct white_for_table *)malloc(sizeof(struct white_for_table));
    if (v == NULL){
      printf("error: in new white for table\n");
      exit(1);
    }
    v->previous = v->next = NULL;
    v->status = NEW;
    v->white = wblock;
    return v;
}

/*****************************************************************/
/*                                                               */
/*  sort white block in x order                                  */
/*                                                               */
/*****************************************************************/
struct whiteblock **sort_x_white(headblock, numblock)
struct block_rect *headblock;
int
*numblock;
{
    int   i, j, stop;
    struct whiteblock *tmpw, **blockx;

    *numblock = 0;
    for (tmpw = headblock->firstwhite; tmpw != NULL; tmpw = tmpw->next)
        (*numblock)++;

    blockx = (struct whiteblock **) malloc((*numblock)*
                                    sizeof(struct whiteblock *));
    if (blockx == NULL){
      printf("allocation error: in sort_x_white\n");
      exit(1);
    }

    for (tmpw = headblock->firstwhite, i =0; tmpw != NULL; tmpw = tmpw->next, i++)
        blockx[i] = tmpw;

    for (i = 0; i < *numblock ; i++){
      for (j = 0, stop = 0; j < *numblock -1 -i; j++)
        if (blockx[j]->mod_upperx > blockx[j+1]->mod_upperx){
            tmpw  = blockx[j+1];
            blockx[j+1] = blockx[j];
            blockx[j] = tmpw;
            stop = 1;
        }
      if (stop == 0) break;
    }

    return blockx;
}

/*****************************************************************/
/*                                                               */
/*  sort white block in x order                                  */
/*                                                               */
/*****************************************************************/
struct whiteblock **sort_y_white(headblock, numblock)
struct block_rect *headblock;
int
*numblock;
{
    int   i, j, stop;
    struct whiteblock *tmpw, **blocky;

    *numblock = 0;
    for (tmpw = headblock->firstwhite; tmpw != NULL; tmpw = tmpw->next)
        (*numblock)++;

    blocky = (struct whiteblock **) malloc((*numblock)*
                                    sizeof(struct whiteblock *));
    if (blocky == NULL){
      printf("allocation error: in sort_x_white\n");
      exit(1);
    }

    for (tmpw = headblock->firstwhite, i =0; tmpw != NULL; tmpw = tmpw->next, i++)
        blocky[i] = tmpw;

    for (i = 0; i < *numblock ; i++){
      for (j = 0, stop = 0; j < *numblock -1 -i; j++)
        if (blocky[j]->mod_uppery > blocky[j+1]->mod_uppery){
            tmpw  = blocky[j+1];
            blocky[j+1] = blocky[j];
            blocky[j] = tmpw;
            stop = 1;
        }
      if (stop == 0) break;
    }

    return blocky;
}

/*****************************************************************/
/*                                                               */
/*  first two top right_x                                        */
/*                                                               */
/*****************************************************************/
void first_x_twotop(wtable, top1, top2)
struct white_for_table *wtable;
int
*top1, *top2;
{
    struct white_for_table **tmpw, *tmp
    int   num, stop, i, j;

    for (tmp = wtable, num = 0; tmp != NULL; tmp = tmp->next, num++);
    tmpw = (struct white_for_table **)malloc(num*
```

【図１２０】

```
                    sizeof(struct white_for_table*));
    for (tmp = wtable, num = 0; tmp != NULL; tmp = tmp->next, num++)
        tmpv[num] = tmp;

    for ( i = 0; i < num ; i++){
        for (j = 0, stop = 0; j < num -1 -i; j++)
            if (tmpv[j]->white->mod_rightx> tmpv[j+1]->white->mod_rightx){
                tmp = tmpv[j+1];
                tmpv[j+1] = tmpv[j];
                tmpv[j] = tmp;
                stop = 1;
            }
        if (stop == 0) break;
    }
    *top2 = -1;
    *top1 = tmpv[0]->white->mod_rightx;
    for (i = 1; i < num ; i++){
        if (tmpv[i]->white->mod_rightx <= tmpv[0]->white->mod_rightx+OFFSET){
            if (tmpv[i]->white->mod_rightx > *top1)
                *top1 = tmpv[i]->white->mod_rightx;
        } else {
            *top2 = tmpv[i]->white->mod_rightx;
            break;
        }
    }
    if (*top2 == -1) *top2 = *top1;
}

/*****************************************************************************/
/*                                                                           */
/*      first_h_protect( wtable, top1, top2)                                 */
/*                                                                           */
/*****************************************************************************/
void first_h_protect( wtable, top1, top2)
struct white_for_table *wtable;
int   *top1, *top2;
{
    struct white_for_table **tmpv, *tmp;
    int   num, stop, i, j;

    for (tmp = wtable, num = 0; tmp != NULL; tmp = tmp->next, num++);
    tmpv = (struct white_for_table **)malloc(num*
                sizeof(struct white_for_table*));
    for (tmp = wtable, num = 0; tmp != NULL; tmp = tmp->next, num++)
        tmpv[num] = tmp;

    for ( i = 0; i < num ; i++){
        for (j = 0, stop = 0; j < num -1 -i; j++)
            if (tmpv[j]->white->mod_lowery> tmpv[j+1]->white->mod_lowery){
                tmp = tmpv[j+1];
                tmpv[j+1] = tmpv[j];
                tmpv[j] = tmp;
                stop = 1;
            }
        if (stop == 0) break;
    }
    *top2 = -1;
    *top1 = tmpv[0]->white->mod_lowery;
    for (i = 1; i < num; i++){
        if (tmpv[i]->white->mod_lowery <= tmpv[0]->white->mod_lowery+OFFSET){
            if (tmpv[i]->white->mod_lowery > *top1)
                *top1 = tmpv[i]->white->mod_lowery;
        } else {
            *top2 = tmpv[i]->white->mod_lowery;
            break;
        }
    }
}

/*****************************************************************************/
/*                                                                           */
/*      copy_white( white1, white2)                                          */
/*                                                                           */
/*****************************************************************************/
void copy_white(white1, white2)
struct whiteblock *white1, *white2;
{
    white1->x = white2->x;
    white1->y = white2->y;
    white1->upper_y = white2->upper_y;
    white1->lower_y = white2->lower_y;
    white1->left_x = white2->left_x;
    white1->right_x = white2->right_x;
    white1->mod_uppery = white2->mod_uppery;
    white1->mod_lowery = white2->mod_lowery;
    white1->mod_leftx = white2->mod_leftx;
    white1->mod_rightx = white2->mod_rightx;
    white1->width = white2->width;
    white1->line_with_black = white2->line_with_black;
    white1->pairright = white2->pairright;
    white1->pairleft = white2->pairleft;
    white1->index = white2->index;
    white1->pairright = white2->pairleft = NULL;
}

/*****************************************************************************/
/*                                                                           */
/*      combine_white(nblock, white1, white2)                                */
/*                                                                           */
/*****************************************************************************/
void combine_white(nblock, white1, white2)
struct block_fact *nblock;
struct whiteblock *white1, *white2;
{
    struct whiteblock *tmpwhite;
    int   i, j, total;

    if (white1->numsubwhite == 0){
        total = white2->numsubwhite+1;
        if (white2->numsubwhite == 0) total++;
        white1->subwhite = new_white(0,total-1);
        copy_white(white1->subwhite, white1);
    } else {
        total = white1->numsubwhite+white2->numsubwhite;
        if (white2->numsubwhite == 0) total++;
        tmpwhite = white1->subwhite;
        white1->subwhite = new_white(0,total-1);
        for (i = 0; i <= white1->numsubwhite-1; i++)
            copy_white(&(white1->subwhite[i]), &(tmpwhite[i]));
        free((struct whiteblock *)tmpwhite);
    }
    if (white2->numsubwhite == 0) {
        copy_white(&(white1->subwhite[total-1]), white2);
    } else {
        tmpwhite = white2->subwhite;
        for (i = total-1, j= white1->numsubwhite-1; j>=0 ; i--, j--)
            copy_white(&(white1->subwhite[i]), &(tmpwhite[j]));
        free((struct whiteblock *)tmpwhite);
    }
    white1->numsubwhite = total;
    if (white2->mod_uppery < white1->mod_uppery)
        white1->mod_uppery = white2->mod_uppery;
```

【図121】

```
if (white2->mod_lowery > white1->mod_lowery)
    white1->mod_lowery = white2->mod_lowery;
if (white2->mod_leftx < white1->mod_leftx)
    white1->mod_leftx = white2->mod_leftx;
if (white2->mod_rightx > white1->mod_rightx)
    white1->mod_rightx = white2->mod_rightx;
white1->left_x = white1->mod_leftx;
white1->right_x = white1->mod_rightx;
white1->upper_y = white1->mod_uppery;
white1->lower_y = white1->mod_lowery;

if (nblock->firstwhite == white2){
    nblock->firstwhite = white2->next;
    if (nblock->firstwhite != NULL) nblock->firstwhite->previous = NULL;
    else nblock->currentwhite = NULL;
} else if (nblock->currentwhite == white2){
        nblock->currentwhite = white2->previous;
        nblock->currentwhite->next = NULL;
    } else {
        white2->previous->next = white2->next;
        white2->next->previous = white2->previous;
    }
    free((struct whiteblock *) white2);
}

/*****************************************************************************/
/*                                                                           */
/*********  test table in vertical direction                        **********/
/*                                                                           */
/*****************************************************************************/
void test_vertical(nblock)
struct block_rec *nblock;
{

    int  top1, top2, num_v, combine, stop, replace, max;
    struct whiteblock *a_white, *tmp;
    struct white_for_table *head, *tmp1, *tmp2;

    top1 = nblock->upper_y;
    top2 = nblock->lower_y;
    replace = NEW; stop = MONITOR;
  /*
printf("get into vertical\n");
  */
  /*
for (tmp = nblock->firstwhite; tmp != NULL; tmp = tmp->next)
    printf("%d %d %d %d %d %d\n", tmp->index,
            tmp->mod_leftx, tmp->mod_uppery, tmp->mod_rightx, tmp->mod_lowery);
printf("top1 = %d top2 = %d \n", top1, top2);
  */
    a_white = sort_y_white(nblock, &num);
  /*
for ( v = 0; v < num; v++)
    printf("%d %d %d %d %d %d %d \n", a_white[v]->index,
            a_white[v]->left_x, a_white[v]->upper_y, a_white[v]->right_x,
            a_white[v]->lower_y );
  */
    head = new_table(a_white[0]);
    for (v = 1; v < num); v++){
        if (a_white[v]->mod_uppery < top2) {
  /*
    printf("%d %d %d %d %d %d %d \n", a_white[v]->index,
            a_white[v]->mod_leftx, a_white[v]->mod_uppery, a_white[v]->mod_rightx,
            a_white[v]->mod_lowery);
  */
            for (tmp2 = head; tmp2 != NULL; tmp2 = tmp2->next){
  /*
printf("in spot1\n");
  */
                if (tmp2->next == NULL){
                    tmp2->next = new_table(a_white[v]);
                    tmp2->next->previous = tmp2;
                    tmp2->previous = tmp2;
                    replace = NEW;
                    break;
                } else if (a_white[v]->mod_leftx >=
                           tmp2->white->mod_rightx-OFFSET) {
  /*
printf("in spot2\n");
  */
                    if (tmp2->next == NULL) {
                        tmp3 = new_table(a_white[v]);
                        tmp2->next = tmp3;
                        tmp3->previous = tmp2;
                        replace = NEW;
                        break;
                    } else continue;
                } else {
                    combine = 1;
                    if (tmp2->white->mod_lowery <=
                                a_white[v]->mod_uppery-OFFSET) {
  /*
printf("in spot3\n");
  */
                        if (tmp2->previous != NULL) {
                            if (((tmp2->previous->white->mod_lowery
                                >= tmp2->white->mod_lowery-OFFSET)&&
                                (tmp2->previous->white->mod_lowery
                                <= tmp2->white->mod_lowery+OFFSET))||
                                ((tmp2->previous->white->mod_uppery
                                >= tmp2->white->mod_uppery-OFFSET)&&
                                (tmp2->previous->white->mod_uppery
                                <= tmp2->white->mod_lowery+OFFSET)))
                                combine = 0;
                        }
                        if ((combine == 1)&&(tmp2->next != NULL)){
                            if (((tmp2->next->white->mod_lowery
                                <= tmp2->white->mod_lowery+OFFSET)&&
                                (tmp2->next->white->mod_lowery
                                >= tmp2->white->mod_lowery-OFFSET))||
                                ((tmp2->next->white->mod_uppery
                                >= tmp2->white->mod_lowery-OFFSET)&&
                                (tmp2->next->white->mod_uppery
                                <= tmp2->white->mod_lowery+OFFSET)))
                                combine = 0;
                        }
                        if ((combine == 1)&&
                            ((tmp2->white->mod_rightx >
                                a_white[v]->mod_leftx-OFFSET))&&
                            (tmp2->white->mod_leftx <
                                a_white[v]->mod_rightx+OFFSET))
                            combine = 0;
                    }
                    if (combine == 1) {
  /*
printf("in spot4\n");
  */
```

【図122】

```
                    combine_white(nblock,tmpw->white,s_white[w]
                    tmpw->status = NEW;
                    replace = NEW;
                    break;
                } else {
                  if(tmpw->status == OLD){
/* printf('in spot5\n'); */
                    tmpw->white = s_white[w];
                    tmpw->status = NEW;
                    replace = NEW;

                    if(s_white[w]->mod_uppery>top1+OFFSET)
                        s_white[w]->mod_uppery = top1;

                    for (tmpw = tmpw->next; tmpw !=NULL;){
                        if((tmpw->white->mod_rightx <=
                          s_white[w]->mod_rightx+OFFSET)){
                          if (tmpw->white->mod_lowery >
                              s_white[w]->mod_uppery+OFFSET)
                            combine_white(nblock,
                                s_white[w], tmpw->white);
                          tmpw->previous->next = tmpw->next;
                          if (tmpw->next != NULL)
                            tmpw->next->previous =
                                tmpw->previous;
                          tmpw2 = tmpw->next;
                          free((struct white_for_table *)
                              tmpw);
                          tmpw = tmpw2;
                        } else break;
                    }
                  } else {
/* printf('in spot6\n');  */
                            stop = STOP;
                        }
                    break;}
            }
          }
          } else stop = STOP;
          if (stop == STOP) {
              v--;
              if (replace == NONE){
/* printf('in spot7\n'); */
                  for ( tmpw = head;  tmpw != NULL; tmpw = tmpw->next;
                    if (tmpw->white->mod_lowery < top2;
                        tmpw->white->mod_lowery = top2;
              }
/* printf('in spot8\n'); */
            for (tmpw=head;(tmpw != NULL)&&(tmpw->next != NULL);){
/*
printf('tmpw = %d x = %d y = %d x2 = %d y2 = %d \n', tmpw->white->index,
        tmpw->white->mod_leftx, tmpw->white->mod_uppery, tmpw->white->mod_rightx,
        tmpw->white->mod_lowery);
*/
                tmpw->status = OLD;
                if (tmpw->white == tmpw->next->white) {
                  tmpw2 = tmpw->next;
                  tmpw->next = tmpw2->next;
                  if (tmpw2->next != NULL)
                    tmpw2->next->previous = tmpw;
                  free((struct white_for_table *) tmpw2);
                } else tmpw = tmpw->next;

            }
            tmpw->status = OLD;
/*
```

```
printf('tmpw = %d x = %d y = %d x2 = %d y2 = %d \n', tmpw->white->index,
                tmpw->white->mod_leftx, tmpw->white->mod_uppery, tmpw->white->mod_rightx,
                tmpw->white->mod_lowery);
printf('in spot9\n');
*/
                first_y_twotop(head, &top1, &top2);
/* printf('after search top1= %d top2 = %d \n', top1, top2); */
                if (top1 == top2) top2 = nblock->lower_y;
                stop = NONSTOP; replace = NONE;
            }
        }
        max = head->white->mod_lowery;
        for ( tmpw = head;  tmpw != NULL; tmpw = tmpw->next)
            if (tmpw->white->mod_lowery > max)
                max = tmpw->white->mod_lowery;
        for ( tmpw = head,  tmpw != NULL; tmpw = tmpw->next)
            if (tmpw->white->mod_lowery < max)
                tmpw->white->mod_lowery = max;
for (tmp = nblock->firstwhite; tmp != NULL; tmp = tmp->next){
/*  printf('%d x1 = %d y1 = %d x2 = %d y2 = %d \n', tmp->index,
            tmp->mod_leftx, tmp->mod_uppery, tmp->mod_rightx, tmp->mod_lowery);
/*
    if (tmp->numsubwhite != 0){
      for (v = 0; w < tmp->numsubwhite; w++)
          printf('   v = %d index = %d x1 = %d y1 = %d x2 = %d y2 = %d \n',
           v, tmp->subwhite[v].index, tmp->subwhite[v].mod_leftx,
           tmp->subwhite[v].mod_uppery, tmp->subwhite[w].mod_rightx,
           tmp->subwhite[w].mod_lowery);
    } else
      printf("%d x1 = %d y1 = %d x2 = %d y2 = %d \n", tmp->index,
              tmp->mod_leftx, tmp->mod_uppery, tmp->mod_rightx, tmp->mod_lowery);

*/
}

/*
printf('get out of vertical\n');
*/
    free((struct whiteblock **)s_white);
}

/*****************************************************************************/
/*********  test table in horizontal  direction                     *********/
/*****************************************************************************/
void test_horizontal(nblock)
struct block_rect *nblock;
{

        int     top1, top2,  num, w, combine, stop, replace, max;
        struct whiteblock **s_white, *tmp;
        struct white_for_table *head;
        struct white_for_table *tmpw, *tmpw2;
/*
printf('get into horizontal\n');
for (tmp = nblock->firstwhite; tmp != NULL; tmp = tmp->next)
    printf('%d x1 = %d y1 = %d x2 = %d y2 = %d \n', tmp->index,
            tmp->mod_leftx, tmp->mod_uppery, tmp->mod_rightx, tmp->mod_lowery);
*/
        top1  = nblock->left_x;
        top2  = nblock->right_x;
        replace = NEW; stop = NONSTOP;

/*  printf('top1 = %d top2 = %d \n', top1, top2); */
        s_white = sort_x_white(nblock, &num);
```

【図123】

【図124】

```
    tmpw->white->mod_lowery));

                tmpw->status = OLD;
                if (tmpw->white == tmpw->next->white) {
                    tmpw2 = tmpw->next;
                    tmpw->next = tmpw2->next;
                    if (tmpw2->next != NULL)
                        tmpw2->next->previous = tmpw;
                    free((struct white_for_table *) tmpw2);
                } else tmpw = tmpw->next;
            }

            tmpw->status = OLD;
        }
/*
printf("tmpw = %d x1 = %d y1 = %d x2 = %d y2 = %d \n", tmpw->white->index,
        tmpw->white->mod_leftx, tmpw->white->mod_uppery, tmpw->white->mod_rightx,
        tmpw->white->mod_lowery));
printf("\n type5\n");
*/
        first_k_tenlop( head, &top1, &top2);
/* printf("after search top1 = %d top2 = %d \n", top1, top2); */
            if (top1 == top2 | top2 = nblock->right_x)
                stop = NONSTOP; replace = NONE;
            }
        }
        max = head->white->mod_rightx;
        for ( tmpw = head; tmpw != NULL; tmpw = tmpw->next)
            if (tmpw->white->mod_rightx > max)
                max = tmpw->white->mod_rightx;
        for ( tmpw = head; tmpw != NULL; tmpw = tmpw->next)
            if (tmpw->white->mod_rightx < max)
                tmpw->white->mod_rightx = max;

    free((struct whiteblock **)s_white));
for (tmp = nblock->firstwhite; tmp != NULL; tmp = tmp->next){
/*  printf("%d x1 = %d y1 = %d x2 = %d y2 = %d \n", tmp->index,
        tmp->mod_leftx, tmp->mod_uppery, tmp->mod_rightx, tmp->mod_lowery);
*/
}
/*
printf("get out of  horizontal\n");
*/
}

/*****************************************************************************/
/*********  table test                                                       */
/*****************************************************************************/

int if_table(nblock)
struct block_rect *nblock;
{
    int original_white, white1, white2;

    struct whiteblock *tmp;

    original_white = nblock->currentwhite->index;

/* first time test */
    test_horizontal(nblock);
    white1 = 0;
    for (tmp=nblock->firstwhite; tmp != NULL; tmp = tmp->next, white1++);
/* printf("original_white = %d  white1  = %d \n",original_white, white1); */
    test_vertical(nblock);
    white2 = 0;
    for (tmp = nblock->firstwhite; tmp  = NULL; tmp = tmp->next, white2++);
/* printf("original_white = %d  white2 = %d \n",original_white, white2);*/

    if ((white2 < 4)||(white2 < original_white/2)) return 2;
    else if ((white2 == original_white)||
             ((white1 == white2)&&(white1 >= 0.8*original_white)))
                return 1;

/* second time test */
    test_horizontal(nblock);
    white1 = 0;
    for (tmp=nblock->firstwhite; tmp != NULL; tmp = tmp->next, white1++);
/* printf("original_white = %d  white1  = %d \n",original_white, white1); */

    if ((white1 < 4)||(white1 < original_white/2)) return 0;
    else if ((white1 == white2)&&(white2 >= 0.8*original_white))
                return 1;

    test_vertical(nblock);
    white2 = 0;
    for (tmp = nblock->firstwhite; tmp != NULL; tmp = tmp->next, white2++);
/* printf("original_white = %d  white2 = %d \n",original_white, white2); */

    if ((white1 == white2)&&(white1 >= 0.8*original_white))
        return 1;
    else
        return 0;
}
```

【図125】

```
/*****************************************************************************/
/*****************************************************************************/
/****************** File: block_util.c                    ********************/
/****************** Author: Shin-Yuan Wang                ********************/
/****************** Date : 1-28-91                        ********************/
/****************** Copyright 1991 Canon Information Systems ******************/
/*****************************************************************************/
/*****************************************************************************/


/*****************************************************************************/
/******** test if covered range overlap                              ********/
/*****************************************************************************/

int if_overlap(a1, a2, b1, b2)
int a1, a2, b1, b2;
{
     if ((a2 < b1)||(b2 < a1)) return 0;
     return 1;
}


/*****************************************************************************/
/********  minimum of two integers                                   ********/
/*****************************************************************************/

int min(a, b)
int a, b;
{
    if (a <= b) return a;
    else return b;
}


/*****************************************************************************/
/********  ratio of overlappint                                      ********/
/*****************************************************************************/

float cover_range(a1, a2, b1,  b2)
int a1, a2, b1, b2;
{
     float range = 0.0;
     int   x1, x2;

     if ((a2 < b1) || (b2 < a1)) return range;
     else {
           x1 = max(a1, b1);
           x2 = min(a2, b2);
          }

     range = (float)(x2-x1+1)/(float)(a2-a1+1);
     return range;
}
```

【図131】

```
/*****************************************************************************/
/*****************************************************************************/
/*********************   Filename: blockline.h                 ***************/
/*********************   Author: Shin-Yuan Wang                ***************/
/*********************   Date:   1-15-92                       ***************/
/****************** Copyright 1992 Canon Information Systems   ***************/
/*****************************************************************************/
/*****************************************************************************/

#define EMPTY       0
#define OVERLAP     1
#define NEXT        2
#define SURROUNDIN  3
#define SURRDUNDOUT 4

#define TH          1.7
#define WIDTHSTD    1.2
#define GAPLENGTH   0

#define DOT_BOTTOM  0.9
#define DOT_WIDTH   1
#define LINEOVERLAP 0.5

#define LEFTGAP     0x10
#define RIGHTGAP    0x01
#define NOISE_NO    3

typedef int fillcondition;

extern    void xs_draw_box();
extern    void convert_headblock();
extern    void transfer_block();
extern    void remove_textblock();
extern    int  max();
extern    int  min();
extern    float cover_range();
extern    struct blockline **new_linepointer();
extern    void sort_linepointer();
```

【図127】

```
/*****************************************************************************/
/*********************   Filename: block_main.h                    ***********/
/*********************   Author: Shin-Yuan Wang                    ***********/
/*********************   Date:   1-15-93                           ***********/
/*****************  Copyright 1992 Canon Information Systems       ***********/
/*****************************************************************************/
/*****************************************************************************/

#define TESTTIME 20
#define TEXT_BLOCK 0x11;
#define LINEPICTURE_BLOCK 0x21
#define HPICTURE_BLOCK    0x31
#define TABLE_BLOCK       0x41
#define LINE_BLOCK        0x51
#define FRAME_BLOCK       0x61
#define UNKNOWN_BLOCK     0x71

#define TEXT_COLOR        0
#define TABLETEXT_COLOR   1
#define LINE_COLOR        2
#define PICTURE_COLOR     3
#define HPICTURE_COLOR    4
#define UNKNOWN_COLOR     5
#define TABLE_COLOR       6
#define FRAME_COLOR       7
#define LINEGAP_COLOR     8

enum {SP_READ = 0x01,
      SP_REDUCE = 0x02,
      SP_CONN = 0x04,
      SEP_BLOCK = 0x08};


struct block_image
{
  int width;
  int height;
  unsigned char** pixel;
};

int xoff[TESTTIME][2] = {{0, 1}, {1, 2}, {0, 1}, {1, 2}, {0, 1},
                         {1, 2}, {0, 0}, {0, 0}, {1, 1}, {1, 1},
                         {2, 2}, {2, 2}, {1, 0}, {2, 3}, {1, 0},
                         {2, 1}, {0, 1}, {0, 1}, {1, 2}, {1, 2}};

int yoff[TESTTIME][2] = {{0, 0}, {0, 0}, {1, 1}, {1, 1}, {2, 2},
                         {2, 2}, {0, 1}, {1, 2}, {0, 1}, {1, 2},
                         {0, 1}, {0, 2}, {0, 1}, {0, 1}, {1, 2},
                         {1, 2}, {1, 2}, {0, 1}, {1, 2}, {0, 1}};

#define  TOTALCOLOR  7

char *color_label[]={"                              Text          ",
                     "                 Text in the table          ",
                     "                    Picture                 ",
                     "               Line Drawing Graphics        ",
                     "  Solid Line (Vertical or Horizontal)       ",
                     "                    Table                   ",
                     "                    Frame                   "};


int color_code[]={TEXT_COLOR, TABLETEXT_COLOR, HPICTURE_COLOR, PICTURE_COLOR,
                  LINE_COLOR, TABLE_COLOR, FRAME_COLOR};
```

【図132】

```
/*****************************************************************************/
/*****************************************************************************/
/*********************   Filename: blocksection.h                  ***********/
/*********************   Author: Shin-Yuan Wang                    ***********/
/*********************   Date:   1-15-93                           ***********/
/*****************  Copyright 1992 Canon Information Systems       ***********/
/*****************************************************************************/
/*****************************************************************************/

#define GAP 16
#define OVERLAP 0.8
#define HOVERLAP 0.5
#define LEVELOVERLAP 0.0
#define LEFTSIDE     0xf0
#define LEFTALIGN    0x80
#define LEFTTOPLONGER 0x40
#define LEFTBOTLONGER 0x20
#define RIGHTSIDE     0x0f
#define RIGHTALIGN    0x08
#define RIGHTTOPLONGER 0x04
#define RIGHTBOTLONGER 0x02
#define TEXTPIXEL      0x08

#define NOISE_COUNT    5

extern int line_inside();
extern int max();
extern int min();
extern int if_overlap();
extern void xs_draw_box();
extern    float cover_range();
void   exit();


struct segment_gap
{  int column_no;
  int *begin_column;
  int *end_column;
};
```

【図128】

```
/*****************************************************************************/
/*                                                                           */
/*                                                                           */
/*                File: blockgeneral.h                                       */
/*                Author: Shih-Yuan Wang                                     */
/*                Date : 1-4-1992                                            */
/*                Copyright 1992 Canon Information Systems                   */
/*                                                                           */
/*                                                                           */
/*****************************************************************************/

#define UNKNOWN 0x8000
#define TEXT 0x01
#define PICTURE 0x02
#define HPICTURE 0x04
#define TABLE 0x08
#define HSOLID 0x10
#define VSOLID 0x20
#define FRAME 0x40
#define TITLE 0x80
#define DOWN 0x100
#define UP 0x200
#define TEXT1 0x401
#define CAPTION 0x800
#define ABSTRACT 0x4000
#define HPOINT 130.0*4.0/72.0
#define ALLPOINT 300
#define MAXTABLE_SPACE 4
#define MAXTABLE_SPACE 0.6
#define EQUAL_RATIO 3
#define STANDARDFONT 12.0*MAXFONT//72.0*SCALE_RATIO
#define CAPTALK 0x40

struct h_outline
{
  int x;
  struct h_outline *next;
};

struct data_outline
{
  int   totalleft;
  int   totalright;
  struct h_outline *maxleft;
  struct h_outline *maxright;
  struct h_outline *current_left;
  struct h_outline *current_right;
};

struct whiteblock
{
  int x, y;
  int right_x, left_x;
  int upper_y, lower_y;
  int mod_rightx, mod_leftx;
  int mod_uppery, mod_lowery;
  int width, length;
  int line_with_black;
  int numsubwhite;
  int index;
  struct whiteblock *previous, *next;
  struct whiteblock *subwhite;
  int   **pairright;
  int   **pairleft;
};

struct block_rect
{
  int   x, y;
  int   right_x, left_x;
  int   upper_y, lower_y;
  int   width, length;
  int   index;
  int   numsubblock;
  int   group;
  int   in_group;
  int   on_point;
  int   att;
  int   cover;
  int   if_sorted;
  int   **pairright;
  int   **pairleft;
  int   avg_height;
  float density;
  struct block_rect *next;
  struct block_rect *previous;
  struct block_rect *firstsubblock;
  struct block_rect *firstnontextblock;
  struct block_rect *currentnontextblock;
  struct block_rect *currentsubblock;
  struct whiteblock *firstwhite, *currentwhite;
};

struct linablock
{
  struct block_rect *first_block;
  struct block_rect *current_block;
  struct linablock  *firstsubline;
  struct linablock  *currentsubline;
  struct linablock  *firstnontextline;
  struct linablock  *currentnontextline;
  struct linablock  *previous;
  struct linablock  *next;

  int   side_gap;
  int   group;
  int   upper_y, lower_y, left_x, right_x;
  int   width, length;
  int   attribute;
  int   numsubblock;
  int   index;
  struct block_rect *under_caption;
  int   l_caption, r_caption;
};

struct sameline
{
  struct sameline *previous;
  struct sameline *next;
  struct linablock *current;
  struct sameline *r_connection;
  struct sameline *l_connection;
  int   index;
};

struct sectionblock
{
  struct linablock   *firstlineblock;
  struct linablock   *currentlineblock;
  struct sectionblock *firstsubsection;
  struct sectionblock *currentsubsection;
  struct sectionblock *firstnontextsection;
  struct sectionblock *currentnontextsection;
```

【図１３０】

```
/*****************************************************************************/
/*                                                                           */
/*  Filename: blockrec.h                                                     */
/*  Author: Shih-Yuan Wang                                                   */
/*  Date:   1-15-92                                                          */
/*  ------- Copyright 1992 Canon Information Systems ------------ Recording  */
/*                                                                           */
/*****************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#define ALLDIRECT       8
#define ONE             0x01
#define ZERO            0

#define LEFT_ONE        0x11
#define RIGHT_ONE       0x21
#define LEFT_ZERO       0x10
#define RIGHT_ZERO      0x20
#define RIGHTLEFT_ZERO  0x30

#define SOLIDFILL       9.0
#define SOLIDDENSITY    0.9
#define STANDARDDEN     0.017453406
#define ROOT            8
#define TEXT_SCALE      2.0
#define VIDEO_RATIO     4
#define GAP             1

#define COVER           -1
#define COVERED         -2
#define ORIGINAL        2

#define CENTER          0
#define NW              1
#define WEST            2
#define SW              3
#define SOUTH           4
#define SE              5
#define EAST            6
#define NE              7
#define NORTH           8

#define SKIP  0
#define LEFT  1
#define RIGHT 2
#define RL    3

typedef int recording;

extern int ifoverlap();
extern ifcable();
extern float covsr_range();

struct loc_point
{ int x,
  int y;} around_center[9]={{0,0}, {-1, -1}, {-1, 0},
                            {-1, 1}, {0, 1}, {1, 1},
                            {1, 0}, {1, -1}, {0, -1}};

int search_map[9][8]={{0,0,0,0,0,0,0,0},
                      {7, NE, NORTH, NW, WEST, SW, SOUTH, SE},
                      {6, NW, WEST, SW, SOUTH, SE, EAST, NE},
                      {5, SW, SOUTH, SE, EAST, NE, NORTH, NW},
                      {4, SE, EAST, NE, NORTH, NW, WEST, SW},
                      {3, NE, NORTH, NW, WEST, SW, SOUTH, SE},
                      {2, EAST, NE, NORTH, NW, WEST, SW, SOUTH},
                      {1, SOUTH, SE, EAST, NE, NORTH, NW, WEST},
                      {0, WEST, SW, SOUTH, SE, EAST, NE, NORTH}};

int map_int[9][8]={
{SKIP, SKIP, SKIP, SKIP, SKIP, SKIP, SKIP, SKIP},
{SKIP, LEFT, LEFT, LEFT, RL, RL, RL, RL},   /*  NORTH_WEST */
{SKIP, LEFT, RIGHT, RIGHT, RIGHT,RIGHT, SKIP, SKIP},  /*  WEST */
{SKIP, SKIP, RIGHT, RIGHT, RIGHT, RIGHT, RL, RL},   /*  SOUTH_WEST */
{SKIP, RIGHT, RIGHT, RIGHT, RIGHT, RL, RL, SKIP},   /*  SOUTH */
{SKIP, RIGHT, RIGHT, RIGHT, RIGHT, RL, RL, RL},   /*  SOUTH_EAST */
{SKIP, RIGHT, LEFT, LEFT, LEFT, LEFT, SKIP, SKIP},   /*  EAST */
{SKIP, SKIP, LEFT, LEFT, LEFT, LEFT, RL, RL},   /*  NORTH_EAST */
{SKIP, LEFT, LEFT, LEFT, LEFT, RL, RL, RL}};  /*  NORTH */
```

【図135】

```
/*****************************************************************************/
/*********************   Filename: table.h                  ******************/
/*********************   Author: Shin-Yuan Wang             ******************/
/*********************   Date:   1-15-92                    ******************/
/*****************  Copyright 1992 Canon Information Systems  ****************/
/*****************************************************************************/
/*****************************************************************************/

#include "blockgeneral.h"

#define NONE 0
#define OFFSET  3
#define OFFSET2 6
#define NEW 1
#define OLD 2
#define NONSTOP 0
#define STOP 1
#define TABLE_TESTTIME 2

struct white_for_table
{
  int    status;
  struct whiteblock *white;
  struct white_for_table *previous, *next;
};
```

【図136】

```
/*****************************************************************************/
/*********************   Filename: images.h                 ******************/
/*********************   Author: Shin-Yuan Wang             ******************/
/*********************   Date:   1-15-92                    ******************/
/*****************  Copyright 1992 Canon Information Systems  ****************/
/*****************************************************************************/
/*****************************************************************************/

#include<stdio.h>
#include<stdlib.h>

struct images
{
  int sizeX;
  int sizeC;
  int dpi;
  unsigned char **pixel;
};
```

【図142】

```
/*
 * Filename: autoinput.h
 * Header file for autoinput.c
 *
 *-------------------------------------------------------------
 */


/* Function Definitions for myxalloc.c */
#ifdef CPP
/* Definition for C++ files */
extern "C" {
  void auto_initialize();
  int auto_input_int();
  void auto_terminate();
}
#else
/* Definition for C files */
void auto_initialize();
int auto_input_int();
void auto_terminate();
#endif
```

【図145】

```
/*-------------------------------------------------------------*/
void Zerovpp_terminate(nzlist)
     struct NonZeroVppList *nzlist; /* The list */
{
  free_ivector(nzlist->list);   /* Deallocate memory */
  nzlist->allocated = 0;        /* Set so won't be accidently used */
  nzlist->size = 0;
}


/*-------------------------------------------------------------*/
void Zerovpp_add(nzlist, a, b)
     struct NonZeroVppList *nzlist; /* The list */
     int a,b;                       /* Start and stop of the zerovpp */
{
  if (nzlist->size+2 > nzlist->allocated) {
    printf("NonZeroVppList FULL!!!\n");
    exit(1);
  }
  nzlist->list[(nzlist->size)++] = a;
  nzlist->list[(nzlist->size)++] = b;
}
```

【図214】

```
/* If your routines are in C++ then CPP must be defined in make file */
#ifdef CPP
   extern "C" void xs_set_foreground(int r, int g, int b);
   extern "C" void xs_set_background(int r, int g, int b);
   extern "C" void xs_init();
   extern "C" void xs_flush();
   extern "C" void xs_plotxy(int x,int y);
   extern "C" void xs_unplotxy(int x,int y);
   extern "C" void xs_redisplay();
   extern "C" void xs_dis_bitmap(int x,int y,int width,int height,char *buf);
   extern "C" void xs_clr_win();
   extern "C" void xs_draw_box(int x,int y,int width,int height);
   extern "C" void xs_string(int x,int y,char *str);
   extern "C" void xs_setcolor(int color);
   extern "C" void xs_win_dim(int *width,int *height);
#endif
```

【図138】

```
CC=     cc
CPP=    cc
COMPILE_PROFILE = -p
COMPILE_STYLE = -O -g
COMPILE_VISIBLE = -DDEBUG_OUTPUT=0VISIBLE

LDLIBS= -lm
LDLIBSX = -L/usr/lib/X11R4 -L/usr/lib/Motif1.1 -lXm -lXt -lX11 -ll -lmalloc -lPW
CPPFLAGS= -I/usr/include/X11R4 -DCPP $(COMPILE_STYLE) $(COMPILE_VISIBLE) $(COMPILE_PR
OFILE)
CFLAGS= -I/usr/include/X11R4 $(COMPILE_STYLE) $(COMPILE_VISIBLE) $(COMPILE_PROFILE)
OBJECTS_multilevel = pagment.o linprocess.o lllha_logic.o outline.o cdab
lle.o zerovpp.o charlist.o cimages.o mymalloc.o xs.o xc.o util.o multilevel2.o recogni_mult.o test_mult.o
lle.o cutdash.o scaleprint.o thlev_file.o autoinput.o
OBJECTS_test = test.o charlist.o mymalloc.o

all: multilevel test

multilevel: $(OBJECTS_multilevel)
        $(CPP) -c multilevel $(OBJECTS_multilevel) $(LDLIBS) $(LDLIBSX)

test: $(OBJECTS_test)
        $(CC) -o test $(OBJECTS_test) $(LDLIBS) $(LDLIBSX)

lllha_logic.o: lllha_logic.C
        $(CPP) -c $(CPPFLAGS) lllha_logic.C

autoinput.o: autoinput.c
        $(CC) -c $(CFLAGS) autoinput.c

combine.o: combine.c
        $(CPP) -c $(CPPFLAGS) combine.C

cimages.o: cimages.C
        $(CPP) -c $(CPPFLAGS) cimages.C

cutdash.o: cutdash.C
        $(CPP) -c $(CPPFLAGS) cutdash.C

linprocess.o: linprocess.C
        $(CPP) -c $(CPPFLAGS) linprocess.C

multilevel.o: multilevel.C
        $(CPP) -c $(CPPFLAGS) multilevel.C

multilevel2.o: multilevel2.C
        $(CPP) -c $(CPPFLAGS) multilevel2.C

mymalloc.o: mymalloc.c
        $(CC) -c $(CFLAGS) mymalloc.c

optimum.o: optimum.c
        $(CC) -c $(CFLAGS) optimum.c

outline.o: outline.C
        $(CPP) -c $(CPPFLAGS) outline.C

pagment.o: pagment.C
        $(CPP) -c $(CPPFLAGS) pagment.C

recognize.o: recognize.c
        $(CC) -c $(CFLAGS) recognize.c

segment.o: segment.C
        $(CPP) -c $(CPPFLAGS) segment.C

charlist.o: charlist.C
        $(CPP) -c $(CPPFLAGS) charlist.C

level3a1.o: level3a1.C
        $(CPP) -c $(CPPFLAGS) level3a1.C

level3a2.o: level3a2.C
        $(CPP) -c $(CPPFLAGS) level3a2.C

scaleprint.o: scaleprint.C
        $(CPP) -c $(CPPFLAGS) scaleprint.C

test.o: test.C
        $(CC) -c $(CFLAGS) test.C

level3a7.o cdab
        $(CPP) -c $(CPPFLAGS) level3a7.C

test_mult.o: test_mult2.C
        $(CPP) -c $(CPPFLAGS) test_mult2.C

thlev_file.o: thlev_file.c
        $(CC) -c $(CFLAGS) thlev_file.c

util.o: util.c
        $(CC) -c $(CFLAGS) util.c

xs.o: xs.C
        $(CC) -c $(CFLAGS) xs.c

xc.o: xc.c
        $(CC) -c -DCPP $(CFLAGS) xc.c

zerovpp.o: zerovpp.C
        $(CPP) -c $(CPPFLAGS) zerovpp.C

charlist.o: charlist.h mymalloc.h
combine.o: charlist.h cimages.h
cimages.o: cimages.h xs.h
level3.o: mymalloc.h
linprocess: charlist.h cimages.h
multilevel.o: charlist.h xs.h mymalloc.h
multilevel2.o: charlist.h cimages.h xs.h mymalloc.h
outline.o: charlist.h cimages.h
pagment.o: cimages.h xs.h mymalloc.h
level3a1.o: charlist.h cimages.h
zerovpp.o: charlist.h cimages.h

rm_visible:
        rm linprocess.o multilevel2.o pagment.o recognize.o scaleprint.o
```

【図139】

```
//
// Filename: 3line_logic.C
// This routine contains optimized 3line logic routines
//
////////////////////////////////////////////////////////////
//
#include"cimages.h"

//.....
// 2x1 OR logic
void t1ine_OR_21(cimages oldline, cimages newline)
{
  int oldr, older;          // Size of old image
  int newr, newc;           // Size of new image
  int i,j,a;
  unsigned char *oc1, *oc2, *nr;   // The old & new column pointers
  unsigned char **or, **nr; // The old & new row pointers

  older = oldline.sizeA;
  oldr = oldline.sizeC;
  newr = (older-1)/2;       // Should be ++
  newc = oldr;
  newline.reallocate(newr+1, newc);
  a = older - newr*2;       // How much is partially covered in the last row
  if (a == 2)
    newr++;                 // It's evenly aligned

  or = oldline.pixel;
  nr = newline.pixel;

  for (i=0; i<newr; i++)    // Do the middle
  {
    oc1 = *or++;
    oc2 = *or++;
    nr = *nr++;
    for (j=0; j<newc; j++)
      *(nr++) = (*(oc1++) | *(oc2++));
  }

  // Do the ends
  nc = *nr;
  if (a == 1)
  {
    oc1 = *or;
    for (j=0; j<newc; j++)
      *(nr++) = *(oc1++);
  }
}

//.....
// 3x1 OR logic
void t1ine_OR_31(cimages oldline, cimages newline)
{
  int older, oldr;          // Size of old image
  int newr, newc;           // Size of new image
  int i,j,a;
  unsigned char *oc1, *oc2, *oc3, *nr; // The old & new column pointers
  unsigned char **or, **nr; // The old & new row pointers

  older = oldline.sizeA;
  oldr = oldline.sizeC;
  newr = (older-1)/2;       // Should be ++
  newc = oldr;
  newline.reallocate(newr+1, newc);
  a = older - newr*2;       // How much is partially covered in the last row
  if (a == 2)
    newr++;                 // It's evenly aligned

  or = oldline.pixel;
  nr = newline.pixel;

  for (i=0; i<newr; i++)    // Do the middle
  {
    oc1 = *or++;
    oc2 = *or++;
    oc3 = *or++;
    for (j=0; j<newc; j++)
      *(nr++) = (*(oc1++) | *(oc2++) | *(oc3++));
  }

  // Do the ends
  nc = *nr;
  if (a == 1)
  {
    oc1 = *or;
    for (j=0; j<newc; j++)
      *(nr++) = *(oc1++);
  }
}

//.....
// 1x1 OR logic
void t1ine_OR_11(cimages oldline, cimages newline)
{
  int older, oldr;          // Size of old image
  int newr, newc;           // Size of new image
  int i,j,a;
  unsigned char *oc1, *oc2, *nr; // The old & new column pointers
  unsigned char **or, **nr; // The old & new row pointers

  older = oldline.sizeA;
  oldr = oldline.sizeC;
  newr = (older-1)/2;       // Should be ++
  newc = oldr;
  newline.reallocate(newr+1, newc);
  a = older - newr*2;       // How much is partially covered in the last row
  if (a == 2)
    newr++;                 // It's evenly aligned

  or = oldline.pixel;
  nr = newline.pixel;

  for (i=0; i<newr; i++)    // Do the middle
  {
    oc1 = *or++;
    oc2 = *or++;
    nr = *nr++;
    for (j=0; j<newc; j++)
      *(nr++) = (*(oc1++) | *(oc2++));
  }

  // Do the ends
  nc = *nr;
  if (a == 1)
  {
    oc1 = *or;
    for (j=0; j<newc; j++)
      *(nr++) = *(oc1++);
  }
}

//.....
// 1x2 AND 2x1 OR Logic
void t1ine_AND_12_OR_21(cimages oldline, cimages newline)
{
  int older, oldr;          // Size of old image
  int newr, newc;           // Size of new image
  int i,j,a,b;
  unsigned char *oc1, *oc2, *nr; // The old & new column pointers
  unsigned char **or, **nr; // The old & new row pointers
  register unsigned char temp1, temp2; // Temporary holding variable
  older = oldline.sizeA;
  oldr = oldline.sizeC;
  newr = (older-1)/2;       // Should be ++
  newc = (oldr-1)/2;        // Should be ++
  newline.reallocate(newr+1, newc+1);

  a = older - newr*2;       // How much is partially covered in the last row
  if (a == 2)
    newr++;                 // It's evenly aligned
  b = older - newr*2;       // How much is partially covered in the last column
  if (b == 2)
  {
    newc++;
    b = 0;                  // == 0 for optimization of ands below
  }
  or = oldline.pixel;
  nr = newline.pixel;

  for (i=0; i<newr; i++)    // Do the middle
  {
    oc1 = *(or++);
```

【図140】

```
oc2 = *(or++);
nc = *(nr++);
for (j=0; j<newc; j++)
{
  temp1 = *(oc1++);
  temp2 = *(oc2++);
  *(nc++) = (temp1 & *(oc1++)) | (temp2 & *(oc2++));
}
if (b)                  // Do the right end
  *(nc) = (*(oc1) | *(oc2));
}

// Do the bottom line (if a == 1)
nc = *nr;
if (a == 1)                     // a == 1  it's one row
{
  oc1 = *or;
  for (j=0; j<newc; j++)
  {
    temp1 = *(oc1++);
    *(nc++) = temp1 & *(oc1++);
  }
  if (b)        // Do the right end
    *(nc) = *(oc1);
}
}

// ..........................................................
// 2x2 AND 1x1 OR Logic
void tline_AND_12_OR_11(cimage& oldline, cimage& newline)
{
  int oldr, oldc;         // Size of old image
  int newr, newc;         // Size of new image
  int i,j,a,b;
  unsigned char *oc1, *oc2, *oc; // The old & new column pointers
  unsigned char **or, **nr;  // The old & new row pointers
  register unsigned char temp1, temp2, temp;

  oldr = oldline.sizer();
  oldc = oldline.sizec();  // Should be >>
  newr = (oldr-1)/2;       // Should be >>
  newc = (oldc-1)/2;
  newline.reallocate(newr+1, newc+1);

  a = oldr - newr*2;    // How much is partially covered in the last row
  if (a == 1)
    newr++;             // It's evenly aligned
  b = oldc - newc*2;    // How much is partially covered in the last column
  if (b == 2)
  {
    newc++;
    b = 0;
  }
  or = oldline.pixel;
  nr = newline.pixel;

  for (i=0; i<newr; i++)        // Do the middle
  {
    oc1 = *(or++);
    oc2 = *(or++);
    oc3 = *(or++);
    nc = *(nr++);
    for (j=0; j<newc; j++)
    {
      temp1 = *(oc1++);
      temp2 = *(oc2++);
      *(nc++) = (temp1 & *(oc1++)) | (temp2 & *(oc2++)) |
                (temp1 & *(oc3++)) | (temp2 & *(oc3++));
    }
    if (b)                  // Do the right end
      *(nc) = *(oc1) | *(oc2) | *(oc3);
  }

  // Do the bottom line (if a == 1 or 2)
  nc = *nr;
  if (a == 1)                   // a == 1  it's one row
  {
    oc1 = *or;
    for (j=0; j<newc; j++)
    {
      temp1 = *(oc1++);
      *(nc++) = temp1 & *(oc1++);
    }
    if (b)      // Do the right end
      *(nc) = *(oc1);
  }
  else {
  if (a == 2)                   // a == 2  it's two rows
  {
    oc1 = *(or++);
    oc2 = *or;
    for (j=0; j<newc; j++)
    {
      temp1 = *(oc1++);
      temp2 = *(oc2++);
      *(nc++) = (temp1 & *(oc1++)) | (temp2 & *(oc2++));
    }
    if (b)      // Do the right end
      *(nc) = *(oc1) | *(oc2);
  }
  }
}
```

【図141】

```
/*
 * Filename: autoinput.h
 * This file contains the code to input from a filename
 *
 *.....
 */

#include <stdio.h>
#include "autoinput.h"

/* Global variables */

int auto_eof_detected;      /* 1= 0 if eof has been detected */
FILE *auto_fp;              /* The File pointer */
int save_to_file;           /* Saving the input to a file? */
int save_count;             /* Number of inputs before newline */
char filename[100];         /* The filename */

/*.....*/
void auto_initialise()
{
  int not_exit = 1;

  auto_eof_detected = 0;

  while(not_exit)
  {
    printf("Input filename for auto-input? ");
    scanf("%s", filename);

    auto_fp = fopen(filename, "r");

    if (auto_fp == NULL)      /* Read open failed */
    {
      auto_fp = fopen(filename, "w");
      if (auto_fp == NULL)    /* Write open failed */
      {
        printf("Cannot open file %s.\n", filename);
                              /* Opened new file */
        printf("%s opened as a new file.\n", filename);
        printf("Enter 1 at the next prompt.\n");
        save_to_file = 1;
        auto_eof_detected = 1;
        not_exit = 0;
      }
    } else {                  /* Read open succeeds */
      printf("Opening existing file %s\n",filename);
      not_exit = 0;
    }
  }

  printf("Do you wish to save the auto-input into this file? (1 = yes)? ");
  save_to_file = auto_input_int(1);
  printf("Warning: Change the int '1' to a '0' to turn off appending to the file\n");
  save_count = 0;
}


int auto_input_int()
{
  int i;

  if (auto_eof_detected)
  {
    printf("\n(Manual Input (-9 = newline, -9999 = exit)) ? ");
    scanf("%d", &i);
    if (i == -9999)
    {
      fclose(auto_fp);
      exit(0);
    }
    if (save_to_file == 1)
    {
      if (i == -9)                /* Print new lines */
      {
        fprintf(auto_fp, "\n"); /* Print the new lines */
        fprintf(auto_fp, "\n");
        fflush(auto_fp);
        save_count = 0;
        return auto_input_int(1); /* And input again */
      }
      fprintf(auto_fp, "%d ", i);
      fflush(auto_fp);
      if (++save_count > 20)
      {
        save_count = 0;
        fprintf(auto_fp, "\n");
      }
      return i;
    }

    if (feof(auto_fp, "%d", &i) == EOF)
    {
      printf("\n***** END OF FILE DETECTED *****\n");
      if (save_to_file == 1)
      {
        fclose(auto_fp);
        auto_fp = fopen(filename, "a");
        if (auto_fp == NULL)
        {
          printf("****** WARNING: CANNOT APPEND TO FILE: %s ******\n", filename);
          save_to_file = 0;
        }
        fprintf(auto_fp, "\n\n");
      }

      auto_eof_detected = 1;
      return auto_input_int(1);
    }

    printf("%d\n",i);
    return i;
  }

  void auto_terminate()
  {
    fclose(auto_fp);
  }
```

【図143】

```
/*
 * Filename: charlist.c
 * This file contains Charlist and NonZeroUpplist utility routines
 *
 *.....
 */

/*.....*/
/*                Routines to handle Character lists                */
/*.....*/
/* By Christopher Sherrick */

#include"charlist.h"
#include"mymalloc.h"

/*.....*/
/* Initialize and allocate a Characterlist (set up curve pointers) */
/* Call this routine once only! */
void Charlist_initialise(clist, clist_max, vector_height)
    struct Charlist *clist;     /* The Characterlist */
    int clist_max;              /* The max number of characters */
    int vector_height;          /* Height of the line (to allocate curves) */
{
    int i;                      /* A looping var */

    /* Allocate memory for characters */
    clist->chr = (struct Character *) malloc((unsigned) clist_max
                                        * sizeof(struct Character));
    if (!clist->chr) {
      printf("Memory Allocation Error in Charlist_initialise()\n");
      exit(1);
    }
    clist->size = 0;            /* Number of characters */
    clist->allocated = clist_max; /* Size Allocated */
    clist->first = 0;           /* No first character */
    clist->vector_height = vector_height;

    /* Allocate memory for curves and set pointers */
    /* Must be optimized, hardcode the array into the code */
    for (i=0; i<clist_max; i++) {
      clist->chr[i].curve1 = ivector(vector_height);
      clist->chr[i].curve2 = ivector(vector_height);
    }
}

/*.....*/
/* Zero a previously allocated Characterlist */
void Charlist_zero(clist)
  struct Charlist *clist;
{
  clist->size = 0;              /* Number of characters */
  clist->first = 0;             /* No first character */
}

/*.....*/
/* Terminate deallocate the Characterlist */
void Charlist_terminate(clist)
  struct Charlist *clist;
{
  int i;

  for (i=clist->allocated-1; i>= 0; i--) { /* Free curves */
    free_ivector(clist->chr[i].curve2);
    free_ivector(clist->chr[i].curve1);
  }

  free((struct Character *) clist->chr);
  clist->size = 0;
  clist->allocated = 0;
  clist->first = 0;
}

/*.....*/
/* Place a new character in the charlist. */
/* Takes the character beyond the charlist ( .chr[size]).
 * calculates .loc, and inserts it into the list */
void Charlist_place_new_char(clist)
  struct Charlist *clist;
{
  struct Character *newchar;    /* Pointer to the new character adding */

  newchar = &(clist->chr[clist->size]); /* Set newchar */

  /* Calculate the position of the new character */
  Charlist_recalculate_loc(newchar);

  /* Check for full list */
  (clist->size)++;              /* Increment the size of the list */
  if (clist->size >= clist->allocated) /* If too big.. */
  {
    printf("Error in Charlist_place_new_char(): Character list is FULL!\n");
    exit(1);
  }

  /* Place new in the list character */
  Charlist_locate_character(clist, newchar);
}

/*.....*/
/* This routine repositions a character in the charlist.  It removes
 * the character, recalculates loc and inserts it in the list accordingly.
 */
void Charlist_replace_char(clist, ch)
    struct Charlist *clist;
    struct Character *ch;
{
  /* Recalculate the position of the character */
  Charlist_recalculate_loc(ch);

  if (clist->size < 2)          /* If it's the only item in the list */
    return;

  /* Remove the character from the list */
  if (ch->prev == 0) {          /* If 1st on list */
    clist->first = clist->first->next;
    clist->first->prev = 0;
  } else {
    ch->prev->next = ch->next;
    if (ch->next != 0)
      ch->next->prev = ch->prev;
  }

  /* Place the character in the list */
  Charlist_locate_character(clist, ch);
}

/*.....*/
/* Insert a character after doing a cut */
```

【図144】

```
/* This routine will insert the character beyond the charlist (:chr[size]).
 * .loc is recalculated for both ch_after and the new character; however
 * .loc is NOT USED TO PLACE THE CHARACTER.  It will
 * place the new character after ch_after!  After finished, it is an
 * extremely good idea to resort the brackets!
 */
void Charlist_insert_after_cut(clist, ch_after)
  struct Charlist *clist;
  struct Character *ch_after;   /* Place the new character after this one */
{
  struct Character *newchar;    /* Pointer to the new character adding */

  newchar = &(clist->chr[clist->size]); /* Set newchar */

  /* Calculate the position of the new character */
  Charlist_recalculate_loc(newchar);
  Charlist_recalculate_loc(ch_after);

  /* Check for full list */
  (clist->size)++;              /* Increment the size of the list */
  if (clist->size >= clist->allocated) /* If too big... */
  {
    printf("Error in Charlist_place_new_char(): Character list is FULL!\n");
    exit(1);
  }

  newchar->prev = ch_after;
  newchar->next = ch_after->next;
  ch_after->next = newchar;
  if (newchar->next != 0)
    newchar->next->prev = newchar;
}

/*.....................................................................*/
/* This routine will remove a character from the character list
 * Note: The memory is NOT reclaimed!!!  So don't rely on this too much.
 * Also, prev and next pointers of ch are NOT altered !!
 */
void Charlist_remove_char(clist, ch)
    struct Charlist *clist;
    struct Character *ch;
{
  /* Remove the character from the list */
  if (ch->prev == 0) {          /* If 1st on list */
    clist->first = clist->first->next;
    clist->first->prev = 0;
  } else {
    ch->prev->next = ch->next;
    if (ch->next != 0)
      ch->next->prev = ch->prev;
  }
}

/*.....................................................................*/
/* This routine will recalculate the position of a character. */
void Charlist_recalculate_loc(ch) /* Output is the position */
    struct Character *ch;       /* The character */
{
  int i,loc;
  /* Calculate the position of the new character */
  loc = 0;
  for (i=ch->cpg; i<=-1; i++) /* Position = average position... */
    loc += ch->curve[i]; /* ... of the curves */
  loc = loc * 4 / (ch->bot - ch->top); /* Set it */
  ch->loc = loc;
}

/*.....................................................................*/
/* This routine will place a character (that is not in the charlist!)
 * into the charlist. */
void Charlist_locate_character(clist, ch)
    struct Charlist *clist;
    struct Character *ch;       /* The character */
{
  int loc;
  struct Character *p;          /* Used for searching for location */
  struct Character *last_p;     /* The character before p */

  /* Is the list empty? */
  if (clist->size == 1)
  {
    ch->next = 0;
    ch->prev = 0;
    clist->first = ch;
    return;
  }

  /* Possible Optimization: Don't start from start of list...
     start from where at */     /* Retrieve position */
  loc = ch->loc;

  /* Is it at the beginning of the list? */
  p = clist->first;             /* Load p with the first item on the list */
  if (loc < p->loc)             /* If it's the first on the list */
  {
    p->prev = ch;
    ch->next = clist->first;    /* Set the pointers */
    ch->prev = 0;
    clist->first = ch;          /* Set the list to this */
    return;
  }

  while (p->loc <= loc)         /* until find record past where to place */
  {
    last_p = p;                 /* Remember last position */
    p = p->next;                /* Get the next character */
    if (p == 0)                 /* If it's the end of list, insert it there */
      break;
  }
  ch->prev = last_p;
  ch->next = last_p->next;      /* Insert new position after last_p */
  last_p->next = ch;
  if (p!=0)                     /* If newchar isn't at the end of the list */
    p->prev = ch;
}

/*.....................................................................*/
/*.....................................................................*/
/*                      Zero Vpp Lists Stuff                           */
/*.....................................................................*/
void Zerovpp_initialize(mallist, size)
    struct MonkeeVpplist *mallist; /* The list */
    int size;                   /* The number of entries (2x allocated!) */
{
  /* Allocate memory for list */
  size = size*2;
  mallist->list = (vector(size)); /* Allocate memory */
  mallist->allocated = size;    /* Set how allocated */
  mallist->size = 0;            /* Set to zero elements */
}
```

【図146】

```
/*
 * Filename: charlist.h
 * Header file for charlist.c
 *
 *.....
 */

/*.....*/
/* ENUM: Chartype: Different types of characters */
/*.....*/
typedef enum {
  CHARACTER_ERASED,
  CHARACTER_ENTER_LINE,
  CHARACTER_LEAD_OFF,
  CHARACTER_JOIN_OUTLINED,
  CHARACTER_COMBINED
Chartype;

typedef enum {
  PASSED_RECOGNITION,
  FAILED_RECOGNITION,
  UNRECOGNIZED
CharRecognition;                    /* Recognition hasn't been tried on it */

/*.....*/
/* Struct Character:
 * Contains information about an outlined character in a page
 */
struct Character {
  Chartype type;                    /* The type of character */
  CharRecognition rec;              /* If it's recognized */
  int level_cut_at;                 /* The level the character was cut at
                                       (for goodcut) */
  int loc;                          /* The location of the character */
  int top;                          /* The start of the character (top)
                                       (inclusive)*/
  int bot;                          /* The bottom of the character (bottom)
                                       (exclusive) */
  int left;                         /* The left most pixel in this character */
  int right;                        /* The right most pixel in this character */
  int *curve1;                      /* The left side of the character(inclusive)
                                       (-1 - not there) */
  int *curve2;                      /* The right side of the character (exclusive)
                                       (-1 - not there) */
  struct Character *next;           /* Ptr to next character in the list
                                       (or 0 if last character) */
  struct Character *prev;           /* Ptr to previous character in the list
                                       (or 0 if first character) */
};

/*.....*/
/* Struct Charlist:
 * Contains information about the list of characters */
struct Charlist {
  struct Character *ch;             /* Array: The characters */
  int size;                         /* Number of characters in the ch[] */
  int allocated;                    /* Number of characters allocated */
  int vector_height;                /* The height of the character curves */
  struct Character *first;          /* The first character in linked list */
};

/*.....*/
/*                    Zero Vpp List Stuff                    */
/*.....*/
/* Struct NonZeroVpplist:
 * Contains information about a list for non-zero Vpp's. */

struct NonZeroVpplist {
  int list;                         /* Array: In : start, 2nd : stop */
  int allocated;                    /* Memory allocated (2,4,6...) */
  int size;                         /* How many used (2,4,6...) */
};

/*.....*/
/*                    Define Routines                        */
/*.....*/

#ifdef CPP

/* Define C++ Routines */
extern "C" {
  void Charlist_initialize(struct Charlist *clist, int clist_max,
                           int vector_height);
  void Charlist_zero(struct Charlist *clist);
  void Charlist_terminate(struct Charlist *clist);
  void Charlist_place_char(struct Charlist *clist, struct Character *ch);
  void Charlist_insert_after_cut(struct Charlist *clist,
  void Charlist_remove_char(struct Charlist *clist, struct Character *ch);
  void Charlist_recalculate_loc(struct Character *ch);
  void Charlist_locate_character(struct Charlist *clist, struct Character *ch);
  void zerovpp_initialize(struct NonZeroVpplist *nzlist, int size);
  void zerovpp_terminate(struct NonZeroVpplist *nzlist);
  void zerovpp_add(struct NonZeroVpplist *nzlist, int a, int b);
}

#else

/* Define F Routines */
void Charlist_initialize();
void Charlist_zero();
void Charlist_terminate();
void Charlist_place_new_char();
void Charlist_replace_char();
void Charlist_insert_after_cut();
void Charlist_remove_char();
void Charlist_recalculate_loc();
void Charlist_locate_character();
void zerovpp_initialize();
void zerovpp_terminate();
void zerovpp_add();

#endif
```

【図147】

```
//
//  Filename:   cimages.cpp
//  This file contains a vast array image utility functions, for a cimage
//
//////////////////////////////////////////////////////////////////////

// undefine this to make images.C run invisibally (except for error)
//#define PRINT_IMAGES_INFO

#include"cimages.h"
#include"xs.h"

//.....................................................................
//== CLASS: cimage   (4D ARRY OF CHAR)
//.....................................................................

cimage::cimage()
{
  sizeR = 0;
  sizeC = 0;
  allocated_sizeR = 0;
  allocated_sizeC = 0;
  dpi = 0;
}

cimage::cimage(int r, int c)
{
  sizeR = 0;                    // So problems won't be caused when
  sizeC = 0;                    //   allocate called deallocate
  allocated_sizeR = 0;
  allocated_sizeC = 0;
  dpi = 0;
  allocate(r,c);
};

cimage::cimage(cimage& img)
{
  allocate(img.sizeR, img.sizeC); // Allocate memory for new cimage (holds sizes)
  for (int i=0; i<sizeR; i++)     // copy old image to new cimage
    for (int j=0; j<sizeC; j++)
      pixel[i][j] = img.pixel[i][j];
};

//////////////////////////////////////////////////////////////////////
// Delete the cimage
cimage::~cimage()
{
  deallocate();
};

//////////////////////////////////////////////////////////////////////
// Allocate memory for an cimage from memory
void cimage::allocate(int r, int c)
{
  deallocate();                 // Remove current cimage
  if (r < 0 || c < 0)
  {
    cerr << "Illegal cimage size: " << r << " by " << c << '\n';
    exit(1);
  };
  sizeR = r;
  sizeC = c;
  if ((r != 0) && (c != 0))     // Set cimage sizes
  {
    pixel = (unsigned char **)malloc((unsigned int) sizeR*sizeof(unsigned char*)); //
Allocate memory
    if (pixel == NULL)  // Allocate memory
    {
      cerr << "Error: Memory Allocation Error (out of memory)\n";
      exit(1);
    }
    for (int i = 0; i<sizeR; i++)
    {
      pixel[i] = (unsigned char *)malloc((unsigned int) sizeC * sizeof(unsigned char)
); // Allocate more mem
      if (pixel[i] == NULL)
      {
        cerr << "Error: Memory Allocation Error (out of memory)\n";
        exit(1);
      }
    }
    allocated_sizeR = sizeR;
    allocated_sizeC = sizeC;
  }
}

//////////////////////////////////////////////////////////////////////
// Deallocate memory for an cimage from memory
void cimage::deallocate()
{
  if ((allocated_sizeR != 0) && (allocated_sizeC != 0))
  {
    for (int i = 0; i < allocated_sizeR; i++)
      free((char *) pixel[i]);
    if (sizeC != 0)
      free((char **) pixel);
    sizeR = 0;                  // Zero out the sizes
    sizeC = 0;
    allocated_sizeR = 0;
    allocated_sizeC = 0;
  }
}

//////////////////////////////////////////////////////////////////////
// This allows allocation of memory from outside the class.  It will
// Allocate memory if it's larger.  If not, it will just adjust the size
void cimage::reallocate(int R, int C)
{
  int new_R, new_C;
  if ((R > allocated_sizeR) || (C > allocated_sizeC))
  {
    new_R = allocated_sizeR;
    new_C = allocated_sizeC;
    if (R > new_R)
      new_R = R;
    if (C > new_C)
      new_C = C;
    allocate(new_R,new_C);      // Re-allocate the image
    printf("---- NOTE: IMAGESIZE REALLOCATED TO [%d, %d]\n", new_R, new_C);
  }
  sizeR = R;
  sizeC = C;
}

//////////////////////////////////////////////////////////////////////
// Make a cimage with a single value, value = value, R and C are the size
void cimage::fill(char value, int R, int C)
```

[図148]

```
{
  allocate(R,C);                    // Make the new cimage (note size adjusted)
  for (int i=0; i<sizeR; i++)
    for (int j=0; j<sizeC; j++)
      pixel[i][j] = value;
};


//////////////////////////////////////////////////////////////////////
// Copy a section of a cimage into another.  image = reference image
// Range is [R1..R2]  [C1..C2]
void cimage::clip(cimage& image, int R1, int R2, int C1, int C2)
{
  int r,c,t;

  if (R2 < R1)
    {t = R1; R1=R2; R2=t;}              // Swap R1 & R2
  if (C2 < C1)
    {t = C1; C1=C2; C2=t;}              // Swap C1 & C2
  if (R1 < 0)  R1 = 0;                  // Check limits
  if (R2 < 0)  R2 = 0;
  if (C1 < 0)  C1 = 0;
  if (C2 < 0)  C2 = 0;
  if (R1 > image.sizeR)  R1 = image.sizeR;
  if (R2 > image.sizeR)  R2 = image.sizeR;
  if (C1 > image.sizeC)  C1 = image.sizeC;
  if (C2 > image.sizeC)  C2 = image.sizeC;

  allocate(R2-R1, C2-C1);

  dpi = image.dpi;
  for (r=0; r<R2-R1; r++)
    for (c=0; c<C2-C1; c++)
      pixel[r][c] = image.pixel[r+R1][c+C1];
}


//////////////////////////////////////////////////////////////////////
//  Clipspace: reduce the size of an image by removing the blank
// space around it.  This adjusts the image size.
void cimage::clipspace()
{
  int R1, R2, C1, C2;   // The starting and stopping rows of new image
  int i, j;
  struct cimage temp;   // Temporary image

  R1 = sizeR;           // Initialize to extremes
  R2 = 0;
  C1 = sizeC;
  C2 = 0;

  for (i=0; i<sizeR; i++)       // Compute dimensions of new image
    for (j=0; j<sizeC; j++)
      if (pixel[i][j] != 0)
      {
        if (i < R1) R1 = i;     // Left edge?
        if (i > R2) R2 = i;     // Right edge?
        if (j < C1) C1 = j;     // Top edge?
        if (j > C2) C2 = j;     // Left edge?
      }

  if (R1 > R2) R1 = R2 = 0;     // Check for no pixel found
  if (C1 > C2) C1 = C2 = 0;

  temp.allocate(R2-R1+1, C2-C1+1);      // New image is this size
  for (i=R1; i<=R2; i++)                // Copy clipped image
    for (j=C1; j<=C2; j++)
      temp.pixel[i-R1][j-C1] = pixel[i][j];

  allocate(temp.sizeR, temp.sizeC);     // Copy temp image to current image
  for (i=0; i<sizeR; i++)
    for (j=0; j<sizeC; j++)
      pixel[i][j] = temp.pixel[i][j];
}


//////////////////////////////////////////////////////////////////////
// Read in part of a Tiff file
void cimage::readtiff(char *filename)
{
  FILE *tagfile;                // Input file
  long int tagfile_position;    // The our file position (in bytes)
  long int width;               // Width of picture
  long int length;              // Length of picture
  long int black;               // Value of Black pixel
  long int bps;                 // Bytes per strip
  long int raster_offset;       // Start of image data
  long int numtags, tag, type, len, value;      // Tag values
  long int i,j;

  if ((tagfile = fopen(filename, "rb")) == NULL)
  {
    cerr << "\nError in opening " << filename << "\n";
    exit(1);
  }
  tagfile_position = 0;

  // Read in header and make sure it's Intel!
  i = get2bytes(tagfile,tagfile_position);
  if (i != 0x4949)
  {
    cout << "... Error: TIFF File not in Intel Format\n";
    exit(1);
  }

  // Read in Magic number
  i = get2bytes(tagfile,tagfile_position);
  if (i != 0x002A)
  {
    cout << " BAD Magic Number for TIFF file!!!\n";
    exit(1);
  }

  // Find First File directory
  j = get4bytes(tagfile,tagfile_position);
  while(tagfile_position < j)
  {
    i = get2bytes(tagfile,tagfile_position);
  }

  // Tags in Directory
  numtags = get2bytes(tagfile,tagfile_position);

  while (--numtags >= 0) {
    tag = get2bytes(tagfile, tagfile_position);
    type = get2bytes(tagfile, tagfile_position);
    len = get4bytes(tagfile, tagfile_position);
    value = get4bytes(tagfile, tagfile_position);

    if ((len != 1) && ((tag & 32768) == 0))
    {
      cout << "Error: Tag Length != 1!\n";
```

【図149】

```
        exit(1);
    }

    switch (tag)
    {
      case 0x00FE:          // NewSubfileType
        if (value != 0)
        {
          cout << "Can't Deal with Non-Standard NewSubFileType\n";
          exit(1);
        }
        break;
      case 0x00FF:          // NewSubfileType
        if (value != 1)
        {
          cout << "Subfile Type not full resolution\n";
          exit(1);
        }
        break;
      case 0x0100:          // Width
        width = value;
        break;
      case 0x0101:          // Length
        length = value;
        break;
      case 0x0102:          // Bits per sample
        if (value != 1)
        {
          cout << ": Error: Bits/Sample must be 1";
          exit(1);
        }
        break;
      case 0x0103:          // Compression
        if (value != 1)
        {
          cout << "Compression Used Error!\n";
          exit(1);
        }
        break;
      case 0x0106:          // Black is
        black = value;
        break;
      case 0x0107:          // Thresholding tag
        if ((value < 1) || (value > 3))
        {
          cout << "Thresholding Tag = Illegal value!\n";
          exit(1);
        }
        break;
      case 0x0111:          // Offset to Raster Data
        raster_offset = value;
        break;
      case 0x0117:
        bps = value;
        break;
      case 0x011a:
        break;
      case 0x011b:
        break;
      case 0x0128:
        break;
      default:
        if ((value & 32768) == 0) {
          cout << "0000 Warning: An unsupported tag appears in the TIF file\n";
        }
        break;
    } // switch
  } // while

  while(tagfile_position < raster_offset)
  {
    i = get2bytes(tagfile,tagfile_position);
  }

  //-------------------------  READ IN FILE  -----------------------------
  block=(int(*)[int] length, (int) width);

  int v;

  for (i = 0; i<length; i++)
  {
    for (j = 0; j<width; j++)
    {
      if ((j&8) == 0)
      {
        v = getc(tagfile);
        tagfile_position++;
        if (v == EOF)
        {
          cerr << "\n Error: Premature End of File found in tiff file\n";
          exit(1);
        }
      }
      if (v & 128) == black*128)
        pixel[i][j] = 0;
      else
        pixel[i][j] = 1;
      v = v << 1;
    }
  }

  fclose(tagfile);
}

//////////////////////////////////////////////////////////////////////////////
// Read in a byte.  File position is updated
// eoferror = 1 to error on EOF, 0 not
int cimage::getbyte(FILE *f, long int& file_position, int eoferror)
{
  int c1;
  c1 = getc(f);
  if ((c1 == EOF) && (eoferror == 1))
  {
    cerr << "\n Error: Premature End of File (in getbyte)\n";
    exit(1);
  }
  file_position++;
  return c1;
}

//////////////////////////////////////////////////////////////////////////////
// Read in an integer from a file.  [LSB, MSB]
long int cimage::get2bytes(FILE *f, long int& fp)
{
  int c1, c2;
  c1 = getbyte(f, fp, 1);
  c2 = getbyte(f, fp, 1);
  return c1 + c2*256;
}

//////////////////////////////////////////////////////////////////////////////
// Read in a long integer
long int cimage::getlbytes(FILE *f, long int& fp)
{
```

【図150】

```
  long int c1, c2;
  c1 = get2bytes(f, fp);
  c2 = get2bytes(f, fp);
  return c1 + c2*65536;
}


////////////////////////////////////////////////////////////////////
// Ploc image using Flash Graphics at (xloc, yloc) with color
void cimage::plot(int x, int y)
{
  int px,py;              // Pixel coords on the screen (upper left)
  char color;

  for (py=0; py<sizeR; py++)
    for (px=0; px<sizeC; px++)
      {
        color = pixel[py][px];
        if (color & 1)
          xs_plotxy(x+px, y+py); // Draw it
      }
};


////////////////////////////////////////////////////////////////////
// Plot image with a box around it (with color 1)
void cimage::plotbox(int x, int y)
{
  int px,py;

  for (px=0; px<sizeC+2; px++) {        // Draw horizontal lines
    if ((x+px+y)%2 == 0)
      xs_plotxy(x+px, y);
    if ((x+px+y+sizeR+1)%2 == 0)
      xs_plotxy(x+px, y+sizeR+1);
  }
  for (py=1; py<sizeR+1; py++) {
    if ((x+y+py)%2 == 0)
      xs_plotxy(x, y+py);
    if ((x+sizeC+1+y+py)%2 == 0)
      xs_plotxy(x+sizeC+1, y+py);
  }
  plot(x+1, y+1);       // Plot the image
}
```

【図151】

```
//
//  Filename:   cimages.h
//  This is the header file for cimages.C
//
////////////////////////////////////////////////////////////////////
//
#define KEN_DEBUG 1
#include<stream.h>
#include<stdio.h>
#include<stdlib.h>
#include<math.h>

class gimage;           // Forward reference

//**************************************************************************
//*** CLASS: cimage
//**************************************************************************
class cimage
{
public:
  int           sizeR;          // Number of rows (first index of array)
  int           sizeC;          // Number of cols. (second index of array)
  int           allocated_sizeR; // Amount malloced
  int           allocated_sizeC; // Amount malloced
  int           dpi;            // Dots per inch of the image (0 if unknown)
  unsigned char **pixel;        // The binary image (2d array)

  cimage();                     // Initialize
  ~cimage();                    // Destroy
  cimage(int r, int c);         // Initialize with a size
  cimage(cimage& temp);

  void reallocate(int R, int C); // Will allocate memory for an image
  void fill(char value, int R, int C); // Fill image with a constant value
  void clip(cimage& image, int R1, int R2, int C1, int C2);
  void clipspace();                     // Clips zero space surrounding image
  void readtiff(char *filename);  // Read in a tiff image
  void plot(int x, int y);              // Plot binary image
  void plotbox(int x, int y); // Plot with a box around it

private:
  void allocate(int r, int c);       // Allocate memory
  void deallocate();                 // Deallocate memory
  int getibyte(FILE *f, long int& file_position, int eoferror);
  long int get2bytes(FILE *f, long int& fp);
  long int get4bytes(FILE *f, long int& fp);
};
```

【図167】

```
  extern void cut_dash(cimage& line, struct Charlist* clist);
  cut_dash((*line_to_process), clist);
#endif

#ifdef COMBINING
  charlist_combine((*line_to_process), clist);
#endif

#ifdef PRINT_FINAL_CHARLIST
  ///////// Print the characterlist
  xs_clr_win();
  print_charlist((*line_to_process), clist, 10, 10);
  xs_flush();
#endif

#ifdef DISPLAY
  xs_flush();
#endif
  printf("Displaying Final Segmentation.  Enter 0 to continue? ");
  int dummy;
  dummy = auto_input_int();

  zerovpp_terminate(&masterallist);     // *** DELETE ****

}
```

[図152]

```
//
// Filename: combine.C
// Routine for combining incorrectly seperated characters
//
////////////////////////////////////////////////////////////////
//

#include"charlist.h"
#include"cimages.h"
#include"xs.h"                  // Just for debugging!!  DELETE EVENTUALLY

#define VERY_SMALL -32000       // A very small integer (for image size)

#define FRAC_VALID_DIST 9.0     // How far apart breaks must be (in 8x pixels
                                // count) to be overlapping.  Multiply this
                                // By the smaller dot to get range.
#define DOT_TO_BODY_HEIGHT_IJ 2 // How much longer the body of the i must be
                                // than the dot (body > X x dot)
#define PERCENT_TYPE1_RATO 0.75 // The ratio of the dot the line should
#define PERCENT_TYPE1_RMIN 0.24 // Minimum ratio of dot width / line width
#define PERCENT_TYPE1_RMAX 0.77 // Maximum ratio of dot width / line width
#define PCNT_TO_TAKE_SLOPE 0.1  // Ignore top and bottom 10% for computing the
                                // slope of the line in a percent sign.
#define PCNT_PERCENT_OVERLAP 0.25 // Multiply this by width of line of % to see
                                // How far the characters must overlap
#define PCNT_DOT_WIDTH_MIN 0.50 // Minimum ratio of dot width / line width
#define PCNT_DOT_WIDTH_MAX 1.10 // Maximum ratio of dot width / line width
#define PCNT_DOT_HEIGHT_MIN 0.43 // Minimum ratio of dot height / line height
#define PCNT_DOT_HEIGHT_MAX 0.70 // Maximum ratio of dot height / line height
#define PCNT_MIN_INSLOPE 0.37   // Minimum inverse slope of the line (aka
                                // inverse of maximum slope of line)

// Define a Combinelist which is a list of 4 characters (with information
// stored about them) that are looked at to see if we need to combin them.
struct Combinelist {
  struct Character *c1;         // The first character in the list
  struct Character *c2;         // The second character in the list
  struct Character *c3;         // .. (and so on)  (0 = none)
  struct Character *c4;
  struct Character *next;       // The next character to load in
  int h12;                      // The height difference between 1 & 2
  int h23;                      // (-3 = undefined, 0 = chars overlaps
  int h34;                      // (1 means first higher, 2 means 2nd higher)
  int d12;                      // The horizontal distance between 1 & 2
  int d23;                      // (0=fined from top of 1 to bottom of other)
  int d34;                      // (VERY_SMALL = undefined)
};

// Define external variables and routines (for displaying and debugging)
extern int cursor_X;
extern int cursor_Y;
extern print_character(cimage& line, struct Character *chr, int x, int y);
extern print_charlist(cimage& line, struct Charlist *clist, int x, int y);


////////////////////////////////////////////////////////////////
//////          Character debugging routines            //////
////////////////////////////////////////////////////////////////
void test_print4(
    cimage4 line,               // Image of the line
    struct Charlist *clist,     // The charlists
    struct Combinelist *comb)   // The list
{
  int x,y;

  x = y = 10;

  xs_clr_win();
  if (comb->c1 != 0)
  {
    print_character(line, comb->c1, x, y);
    x += 10 - comb->c1->left + comb->c1->right;
  }
  if (comb->c2 != 0)
  {
    print_character(line, comb->c2, x, y);
    x += 10 - comb->c2->left + comb->c2->right;
  }
  if (comb->c3 != 0)
  {
    print_character(line, comb->c3, x, y);
    x += 10 - comb->c3->left + comb->c3->right;
  }
  if (comb->c4 != 0)
  {
    print_character(line, comb->c4, x, y);
    x += 10 - comb->c4->left + comb->c4->right;
  }

  print_charlist(line, clist, 10, 20+clist->vector_height);
  xs_flush();
}


////////////////////////////////////////////////////////////////
//////          Character combining routines            //////
////////////////////////////////////////////////////////////////


////////////////////////////////////////////////////////////////
// This routines returns if two characters are above each other.
// Returns 0 for overlapping or character does not exist, 1 for 1st higher,
// 2 for 2nd higher.  It also updates h12.
int comb_height_delta(
  struct Character *c1,         // The first character
  struct Character *c2,         // The second character
  int *h12)                     // the height (updated)
{
  if (*h12 != -1)               // If it's been calculated,
    return (*h12);              // don't recalculate it

  if (c2 == 0)                  // If character doesn't exist
  {
    *h12 = 0;                   // Return no gap exists
    return 0;
  }

  if (c1->bot <= c2->top)       // If bottom of c1 is higher than top of c2
  {
    *h12 = 1;                   // Return the gap
    return 1;
  }

  if (c2->bot <= c1->top)       // If bottom of c2 is higher than top of c1
  {
    *h12 = 2;
    return 2;                   // Return the gap
  }

  *h12 = 0;                     // Return no gap exists
  return 0;
}
```

【図153】

```
    for (i=c2->bot - 4; i < c2->bot; i++)
      botpos += c2->curve2[i] * c2->curve2[i];

  // Compute position and exit
  *dl2 = botpos - toppos;

  f = ((float) *dl2) / ((float) width_small);
  if (f < FRAC_VALID_DIFF)
    return 1;
  else
    return 0;
}


////////////////////////////////////////////////////////////////////////
// This routine is called from comb_dist() with a pair of characters
//that probably isn't an i or j (-1-)
// characters.  this determines if close enough to be combined.
int comb_dist_jj(){
  struct Character *c1,       // The first character
  struct Character *c2,       // The second character
  int hl2,                    // Which character is higher
  int *dl2,                   // The last value
  int width_small)            // The width of the smaller character
{
  int toppos, botpos;         // Positions of the top and bottoms
  int i;
  float f;

  if (hl2 == 1)
  ////// Case when c1 is higher than c2
  {
    // Find bottom pos of c1
    if ((c1->bot - c1->top) < 4)
      botpos = c1->loc;       // If character very small, use .loc
    else
    {
      botpos = 0;
      for (i=c1->bot - 4; i < c1->bot; i++)
        botpos += c1->curve2[i] * c1->curve2[i];
    }

    // Find top pos of c2
    if ((c2->bot - c2->top) < 4)
      toppos = c2->loc;       // If character very small, use .loc
    else
    {
      toppos = 0;
      for (i=c2->top; i < c2->top + 4; i++)
        toppos += c2->curve2[i] * c2->curve2[i];
    }

    // Compute position and exit
    *dl2 = toppos - botpos;

    f = ((float) *dl2) / ((float) width_small);
    if (f < FRAC_VALID_DIFF)
      return 1;
    else
      return 0;
  }

  ////// c2 is higher than c1
  // Find top poi of c1
  if ((c1->bot - c1->top) < 4)
    toppos = c1->loc;         // If character very small, use .loc
  else
  {
    toppos = 0;
    for (i=c1->top; i < c1->top + 4; i++)
      toppos += c1->curve2[i] * c1->curve2[i];
  }

  // Find bottom pos of c2
  if ((c2->bot - c2->top) < 4)
    botpos = c2->loc;         // If character very small, use .loc
  else
  {
    botpos = 0;
```

```
////////////////////////////////////////////////////////////////////////
// This routine is called from comb_dist() with a pair of characters
//that probably isn't an i or j (-1-)
// characters.  this determines if close enough to be combined.
int comb_dist_ji(){
  struct Character *c1,       // The first character
  struct Character *c2,       // The second character
  int hl2,                    // Which character is higher
  int *dl2,                   // The last value
  int width_small)            // The width of the smaller character
{
  int toppos, botpos;         // Positions of the top and bottoms
  int i;
  float f;

  if (hl2 == 1)
  ////// Case when c1 is higher than c2
  {
    // Find bottom pos of c1
    if ((c1->bot - c1->top) < 4)
      botpos = c1->loc;       // If character very small, use .loc
    else
    {
      botpos = 0;
      for (i=c1->bot - 4; i < c1->bot; i++)
        botpos += c1->curve2[i] * c1->curve2[i];
    }

    // Find top pos of c2
    if ((c2->bot - c2->top) < 4)
      toppos = c2->loc;       // If character very small, use .loc
    else
    {
      toppos = 0;
      for (i=c2->top; i < c2->top + 4; i++)
        toppos += c2->curve1[i] * c2->curve1[i];
    }

    // Compute position and exit
    *dl2 = toppos - botpos;

    f = ((float) *dl2) / ((float) width_small);
    if (f < FRAC_VALID_DIFF)
      return 1;
    else
      return 0;
  }

  ////// c2 is higher than c1
  // Find top poi of c1
  if ((c1->bot - c1->top) < 4)
    toppos = c1->loc;         // If character very small, use .loc
  else
  {
    toppos = 0;
```

[図154]

```
else
{
  toppos = 0;
  for (i=c1->top; i < c1->top + 4; i++)
    toppos += c1->curve1[i] + c1->curve2[i];
}

// Find bottom pos of c2
if ((c2->bot - c2->top) < 4)
  botpos = c2->loc;    // If character very small, use .loc
else
{
  botpos = 0;
  for (i=c2->bot - 4; i < c2->bot; i++)
    botpos += c2->curve1[i] + c2->curve2[i];
}

// Compute position and exit
*d12 = botpos - toppos;

f = (((float) *d12) / ((float) width_small));
if (f < FRAC_VALID_DIST)
  return 1;
else
  return 0;
}


////////////////////////////////////////////////////////////////////
// This routines returns how close two non-height-overlapping characters
// are (h12 must be valid!).  Computes characters from top of lower one
// to bottom of higher one.  Returns 0 = Bad, 1 = Good.  Also updates d12
// Position is generated to be relative to dot height
int comb_dist(
  struct Character *c1,          // The first character
  struct Character *c2,          // The second character
  int h12,                       // Which character is higher
  int *d12)                      // The last value

{
  int width_c1;                  // The size of c1 (width)
  int width_c2;                  // The size of c2
  int width_small;               // The size of the smaller character
  int height_c1;                 // The height of c1
  int height_c2;                 // The height of c2
  float f;

  width_c1 = c1->right - c1->left;
  width_c2 = c2->right - c2->left;
  if (width_c1 < width_c2)
    width_small = width_c1;
  else
    width_small = width_c2;

  if (*d12 != VERY_SMALL)        // If it's been computed, don't recompute it
  {
    f = (((float) *d12) / ((float) width_small));
    if (f < FRAC_VALID_DIST)
      return 1;
    else
      return 0;
  }

  height_c1 = c1->bot - c1->top;
  height_c2 = c2->bot - c2->top;

  if (h12 == 1)          // c1 is higher that c2
  {
    if ((height_c1 * DOT_TO_BODY_HEIGHT_IJ) < height_c2)
      return comb_dist_ij(c1, c2, h12, d12, width_small);
    else
      return comb_dist_notij(c1, c2, h12, d12, width_small);
  }
  // Else c2 is higher than c1
  if ((height_c2 * DOT_TO_BODY_HEIGHT_IJ) < height_c1)
  {
    return comb_dist_ij(c1, c2, h12, d12, width_small);
  }
  else
    return comb_dist_notij(c1, c2, h12, d12, width_small);


}


////////////////////////////////////////////////////////////////////
// This routine sees if larger characters are overlapping enough (left and
// right wise.)  used for identifying type 1 percent signs.
// Returns a 1 if overlapping for a type 1 percent, 0 otherwise
// Assumes that the characters ARE overlapping!
int comb_psign_overlap(
  struct Character *c1,          // The first character
  struct Character *c2)          // The second character
{
  int l1, l2;                    // The length of the characters
  int a1, a2;                    // Computed sides of the character
  struct Character *swap;        // Misc var
  int swapi;                     // Misc var

  l1 = c1->right - c1->left;     // Compute widths
  l2 = c2->right - c2->left;

  if (l1 > l2)                   // c1 is longer
  {
    swap = c1;                   // Swap to make c1 shorter
    c1 = c2;
    c2 = swap;
    swapi = l1;
    l1 = l2;
    l2 = swapi;
  }

  if (((float) l1 / (float) l2 < PERCENT_TYPE1_RMIN)
    return 0;
  if (((float) l1 / (float) l2 > PERCENT_TYPE1_RMAX)
    return 0;

  if (c2->left < c1->left)
    a1 = c1->left;
  else
    a1 = c2->left;
  if (c2->right > c1->right)
    a2 = c1->right;
  else
    a2 = c2->right;

  if ((((float) (a2 - a1)) / ((float) l1) > PERCENT_TYPE1_RATO)
    return 1;
  else
    return 0;
}

////////////////////////////////////////////////////////////////////
```

[図155]

```
// This routine will check for the two characters being the 1st
// part of a percent sign o/     Returns 1 for it's a %, 0 for no.
int comb_left_percent(
  struct Character *c1,          // The characters
  struct Character *c2)
{
// Check for o/ part of percent (types 2)

  float h1, w1;                  // The height and widths of the line
  float hd, wd;                  // The height and widths of the dot
  int p;                         // How far from the top to take the slope from

  h1 = (float) (c2->bot - c2->top);
  w1 = (float) (c2->right - c2->left);
  hd = (float) (c1->bot - c1->top);
  wd = (float) (c1->right - c1->left);

  if (w1 * PCNT_PERCENT_OVERLAP <= (float) (c1->right - c2->left))
                                 // If it overlaps enough
  if (w1 * PCNT_DOT_WIDTH_MIN <= wd)
                                 // If dot is width is not too small
  if (w1 * PCNT_DOT_WIDTH_MAX >= wd)
                                 // If dot is width is not too large
  if (h1 * PCNT_DOT_HEIGHT_MIN <= hd)
                                 // If dot is width is not too small
  if (h1 * PCNT_DOT_HEIGHT_MAX >= hd)
                                 // If dot is width is not too large
  {
    p = (int) (w1 * PCNT_TO_TAKE_SLOPE);  // Ignore from top and bottom
    if ((float) (c1->curve1[c2->top+p] - c2->curve1[c2->bot-p]) /
        (float) (c2->bot - c1->top - p - p) >= PCNT_MIN_NSLOPE)
      return 1;
  }
  return 0;
}


//////////////////////////////////////////////////////////////////
// This routine will check for the two characters being the 2nd
// part of a percent sign /o     Returns 1 for it's a %, 0 for no.
int comb_right_percent(
  struct Character *c1,          // The characters
  struct Character *c2)
{
  // Check for o/ part of percent (types 2)

  float h1, w1;                  // The height and widths of the line
  float hd, wd;                  // The height and widths of the dot
  int p;                         // How far from the top to take the slope from

  h1 = (float) (c1->bot - c1->top);
  w1 = (float) (c1->right - c1->left);
  hd = (float) (c2->bot - c2->top);
  wd = (float) (c2->right - c2->left);

  if (w1 * PCNT_PERCENT_OVERLAP <= (float) (c1->right - c2->left))
                                 // If it overlaps enough
  if (w1 * PCNT_DOT_WIDTH_MIN <= wd)
                                 // If dot is width is not too small
  if (w1 * PCNT_DOT_WIDTH_MAX >= wd)
                                 // If dot is width is not too large
  if (h1 * PCNT_DOT_HEIGHT_MIN <= hd)
                                 // If dot is width is not too small
  if (h1 * PCNT_DOT_HEIGHT_MAX >= hd)
                                 // If dot is width is not too large
  {
    p = (int) (w1 * PCNT_TO_TAKE_SLOPE);  // Ignore from top and bottom
    if ((float) (c1->curve2[c1->top+p] - c1->curve2[c1->bot-p]) /
        (float) (c1->bot - c1->top - p - p) >= PCNT_MIN_NSLOPE)
      return 1;
  }
  return 0;
}


//////////////////////////////////////////////////////////////////
// This routine will slide the list over, reading in a new character
// i.e.   1 <- 2 <- 3 <- 4 <- New; (entire list)
void comb_slide(
    struct CombineList *comb)    // The list
{
  comb->c1 = comb->c2;           // Slide the characters down
  comb->c2 = comb->c3;
  comb->c3 = comb->c4;
  comb->c4 = comb->next;
  if (comb->next != 0)
    comb->next = comb->next->next;      // Get the next character in the list
  comb->h12 = comb->h23;         // Slide down heights as well
  comb->h23 = comb->h34;
  comb->h34 = -1;                // Set to undefined
  comb->d12 = comb->d23;         // Slide down distances
  comb->d23 = comb->d34;
  comb->d34 = VERY_SMALL;        // Set to undefined
}


//////////////////////////////////////////////////////////////////
//  This routine will combine two chars (which better be adjacent!)  This
//  works by removing c2 and adding it on to the end of c1.
void comb_combine(
  struct Character *c1,          // The first character
  struct Character *c2)          // The second character
{
  int loc;                       // The location of the new character
  int top;                       // The top of the new character
  int ploc;                      // The location divided by 8 (real position)
  int bot;                       // The bottom of the new character
  int count;                     // Used to compute loc
  int a1,a2;                     // Range of overlapping portion of characters
  int left, right;               // The left and right of new character
  int i;

  // Combine curve1 and curve2, and compute top and bot
  if ((c1->bot < c2->top) || (c2->bot < c1->top))
  {
    /////////////// NonOverlapping Chars ////////////////
    for (i=c2->top; i<c2->bot; i++)     // Copy curves of c2 into c1
    {
      c1->curve1[i] = c2->curve1[i];
      c1->curve2[i] = c2->curve2[i];
    }

    // compute new location
    loc = (c1->loc * (c1->bot - c1->top) + c2->loc * (c2->bot - c2->top)) /
          (c1->bot - c1->top + c2->bot - c2->top);
    ploc = loc/8;

    // Fill blank space between to chars (set to loc, and loc won't change.)
    //   (and set top and bot)
    if (c1->bot < c2->top)
    {
      for (i=c1->bot; i<c2->top; i++)
        c1->curve1[i] = c1->curve1[i] = ploc;
      top = c1->top;
```

【図156】

```
    bot = c2->bot;
  } else {
    for (i=c1->bot; i<c1->top; i++)
      c1->curve1[i] = c1->curve1[i] - ploc;
    top = c2->top;
    bot = c1->bot;
  }
} else {
  ////////////////// Overlapping characters //////////////////
  // Finishes with position in loc, top and bot
  loc = count = 0;
  // deal with upper non-overlapping range
  if (c2->top < c1->top)          // If c2 is higher
  {
    for (i=c2->top; i < c1->top; i++) // then copy upper part of c2 into c1
    {
      c1->curve1[i] = c2->curve1[i];
      c1->curve2[i] = c2->curve2[i];
      loc += c1->curve1[i] - c1->curve2[i];
    }
    count += c1->top - c2->top;
    top = c2->top;
    i1 = c1->top;
  } else {                        // else c1 is higher or equal to c2
    for (i=c1->top; i<c2->top; i++) // compute location info
      loc += c2->curve1[i] - c2->curve2[i];
    count += c2->top - c1->top;
    top = c1->top;
    i1 = c2->top;
  }

  // deal with lower non-overlapping range
  if (c2->bot > c1->bot)          // If c2 is lower
  {
    for (i=c1->bot; i < c2->bot; i++) // Copy lower part of c1 into c1
    {
      c1->curve1[i] = c2->curve1[i];
      c1->curve2[i] = c2->curve2[i];
      loc += c2->curve1[i] - c2->curve2[i];
    }
    count += c2->bot - c1->bot;
    bot = c2->bot;
    i2 = c1->bot;
  } else {                        // else c1 is lower (or equal col c2
    for (i=c2->bot; i < c1->bot; i++)
      loc += c1->curve1[i] - c1->curve2[i];
    count += c1->bot - c2->bot;
    bot = c1->bot;
    i2 = c2->bot;
  }

  // Copy in overlapping range
  for (i=i1; i<i2; i++)
  {
    if (c2->curve1[i] < c1->curve1[i])
      c1->curve1[i] = c2->curve1[i];
    if (c2->curve2[i] > c1->curve2[i])
      c1->curve2[i] = c2->curve2[i];
    loc += c1->curve1[i] - c1->curve2[i];
    count++;
  }

  loc = loc * 4 / count;
}

// Compute left and right
if (c1->left < c2->left)
  left = c1->left;
else
  left = c2->left;
if (c1->right > c2->right)
  right = c1->right;
else
  right = c2->right;

// Load information about the stuff.
c1->loc = loc;
c1->top = top;
c1->bot = bot;
c1->left = left;
c1->right = right;
c1->next = c2->next;              // Skip over c2
if (c2->next != 0)
  (c2->next->prev = c1);
c1->type = CHARACTER_COMBINED;
c2->type = CHARACTER_ERASED;
}

////////////////////////////////////////////////////////////////////
// This routine will combine characters 1 and 2, and slide them
// out of the way.
void comb_combine12(
  struct CombineList *comb)   // The list
{
  comb_combine(comb->c1, comb->c2);
  comb_slide(comb);
  comb_slide(comb);
}

////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////
// This is the main routine to combine a charlist
void charlist_combine(
  struct CharList *clist)     // Image of the line
                              // The character list
{
  struct CombineList comb;    // The 4 characters in the que we're evaluating
  int dummy;

  // Initialize the combination list
  comb.next = clist->first;
  comb_slide(&comb);          // Slide in 4 characters into the list
  comb_slide(&comb);
  comb_slide(&comb);
  comb_slide(&comb);

  // Main loop:  Warning: indentation is not standard! (But not too bad)
  while(comb.c2 != 0)         // Loop until list is empty
  {
    ////////////////// Combine characters over each other
    // Combine 1 & 2?
    if (comb_height_delta(comb.c1, comb.c2, &(comb.h12)))
    if (comb_dist(comb.c1, comb.c2, comb.h12, &(comb.d12)))
    {
      // Combine 2 & 1? instead?
      if (comb.c1 != 0)
      if (comb_height_delta(comb.c2, comb.c1, &(comb.h2)))
      if (comb_dist(comb.c2, comb.c1, comb.h2, &(comb.d2)))
      if (comb.d2 < comb.d1)
      {
```

【図157】

```
// Combine 1&2 and 3&4 instead?
if (comb_height_delta(comb.c3, comb.c4, &(comb.h34)))
if (comb_dist(comb.c3, comb.c4, comb.h34, &(comb.d34)))
if (comb.d34 < comb.d2)
{
  // Combine both 1&2, and 3&4
  comb_combine12(&comb);
  comb_combine34(&comb);
  continue;
}
// Combine chars 3&4
comb_slide(&comb);
comb_combine12(&comb);
continue;

// Combine chars 1&2
comb_combine12(&comb);
continue;

//////////////// Now combine percent signs

if ((comb.c1->right > comb.c2->left) && (comb.c2->right > comb.c1->left))
                    // If overlapping
                    // (Don't do if not overlapping)
{
  // Check for percent sign type 1: (almost horizontal line)
  if (comb_psign_overlap(comb.c1, comb.c2)) // Combine 1 and 2?
  {
    if (comb.c3 != 0)
    if (comb_psign_overlap(comb.c1, comb.c3))   // Combine 1, 2 and 3?
    {
      comb_combine(comb.c2, comb.c3);  // Combine all three characters
      comb_combine(comb.c1, comb.c2);
      comb_slide(&comb);
      comb_slide(&comb);
      comb_slide(&comb);
      continue;
    }
    comb_combine12(&comb);        // Combine c1 & c2
    continue;
  }

  // Check for o/ part of percent (type 2)
  if (comb_left_percent(comb.c1, comb.c2))
  {                               // It's a o/
    if (comb.c3 != 0)
    if (comb_right_percent(comb.c2, comb.c3)) // Check for o/o
    {
      comb_combine(comb.c2, comb.c3);      // Combine all three characters
      comb_combine(comb.c1, comb.c2);
      comb_slide(&comb);
      comb_slide(&comb);
      comb_slide(&comb);
      continue;
    }
    comb_combine12(&comb);                 // Just combine 1 & 2
    continue;
  }

  // Check for /o part of percent
  if (comb_right_percent(comb.c1, comb.c2))
  {
    comb_combine12(&comb);                 // Just combine 1 & 2
    continue;
  }
}
////////////// Don't combine anything
comb_slide(&comb);
}
                                    // Didn't combine anything
```

[図158]

```
//
//  Filename: cutdash.C
//  This file contains the routines to remove the dashes from characters
//  in a charlist
//
////////////////////////////////////////////////////////////////////////
//
//#define PRINT_INFO                    // Print detailed information

////////////////////////////////////////////////////////////////////////
// The includes
#include"charlist.h"
#include"cimages.h"
#include"myalloc.h"

#ifdef PRINT_INFO
#include"xs.h"
extern void print_character(cimage& line, struct Character *ch, int x, int y);
#endif

#define MIN_ABS_WIDTH_TO_CONSIDER 10 // If pixel width less than this, reject.
#define CD_MAX_DASH_HIST 0.3     // x height = max thickness of dash
#define CD_MIN_WIDTH_CONST 0.30 // x height = minimum length of dash
#define CD_MAX_SLOPE_OF_DASH 0.15   // maximum slope of the dash
#define CD_TOO_HIGH 0.2          // x height = max height of dash
#define CD_TOO_LOW 0.8           // x height = max low of dash
                                 //      Note: axis is reversed! (0=up,1=down)

////////////////////////////////////////////////////////////////////////
// External Routines
void l32_cut_character(cimage& line, struct Charlist *clist,
                       struct Character **ch, int p);
void l32_histogram(cimage& line, struct Character **ch, int *hist);


////////////////////////////////////////////////////////////////////////
// Check if there is a dash on the left side
int cd_left_dash_cut(
  cimage& line,                 // The line
  struct Character **ch,        // Pointer to the character we're on (Modified
                                // to skip past newly inserted divisions)
  int *hist)                    // The histogram (previously allocated)
{
  int h;                        // Height of the character
  int t;                        // The thresh crossing point
  int w_min;                    // The minimum width for a dash
  int p1;                       // The break for a left dash
  int w1;                       // The width of the left dash
  int p1, p11, p1h;             // Various points of the dash
  int p2, p21, p2h;             // (See lab log notebook)
  float m;                      // The slope of the dash
  int min_height, max_height;   // The heights of the dashes

  // Compute minimum thickness
  h = (*ch)->bot - (*ch)->top;
  t = (int) (((float) h) * CD_MAX_DASH_HIST);

  // Compute minimum length of dash
  w_min = (int) (((float) h) * CD_MIN_WIDTH_CONST);

  // Compute width of dash
  p1 = (*ch)->left;
  while ((hist[p1] < t) && (p1 < (*ch)->right-1))
    p1++;
  w1 = p1 - (*ch)->left;

#ifdef PRINT_INFO
  printf("Height = %d,  threshold = %d,  Min_dash_length = %d\n", h, t, w_min);
  printf("Left Side:\n");
  printf("  P1 = %d  (W1 = %d)\n", p1, w1);
#endif

  // Reject if width is too short
  if ((w1 < w_min) || (w1 < MIN_ABS_WIDTH_TO_CONSIDER))
  {
    #ifdef PRINT_INFO
    printf("  Rejected: Width of dash is too short.\n");
    #endif
    return 0;
  }

  // Compute p1, p2, p1h, p2h, p11, p21
  p1 = w1/4 + (*ch)->left;              // Find positions of p1 and p2
  p2 = 3*w1/4 + (*ch)->left;
  for (p1h=(*ch)->top; p1h <(*ch)->bot; p1h++)  // Find p1h
    if ((p1 >= (*ch)->curve1[p1h]) && (p1 < (*ch)->curve2[p1h]))
      if (line.pixel(p1h)(p1) & 1)
        break;
  for (p11=(*ch)->bot-1; p11 >= (*ch)->top; p11--)      // Find p11
    if ((p1 >= (*ch)->curve1[p11]) && (p1 < (*ch)->curve2[p11]))
      if (line.pixel(p11)(p1) & 1)
        break;
  for (p2h=(*ch)->top; p2h <(*ch)->bot; p2h++)  // Find p2h
    if ((p2 >= (*ch)->curve1[p2h]) && (p2 < (*ch)->curve2[p2h]))
      if (line.pixel(p2h)(p2) & 1)
        break;
  for (p21=(*ch)->bot-1; p21 >= (*ch)->top; p21--)      // Find p21
    if ((p2 >= (*ch)->curve1[p21]) && (p2 < (*ch)->curve2[p21]))
      if (line.pixel(p21)(p2) & 1)
        break;
  #ifdef PRINT_INFO
  printf("  p1 = %d, p2 = %d\n", p1, p2);
  printf("  p1h, p11  = %d, %d\n", p1h, p11);
  printf("  p2h, p21  = %d, %d\n", p2h, p21);
  #endif

  // Check slope
  m = ((float) (p2h + p21 - p1h - p11))  / ((float) w1);
  if (m < 0)  m = -m;
  #ifdef PRINT_INFO
  printf("  slope = %f\n", m);
  #endif
  if (m > CD_MAX_SLOPE_OF_DASH)
  {
    #ifdef PRINT_INFO
    printf("  Rejected: Dash slopes too much.\n");
    #endif
    return 0;
  }

  // Check height
  max_height = (int) (CD_TOO_HIGH * (float) h) + (*ch)->top;
  min_height = (int) (CD_TOO_LOW * (float) h) + (*ch)->top;

  #ifdef PRINT_INFO
  printf("  Max height = %d, top height = %d\n", max_height, (p1h + p1h)/2);
  #endif
  if ((p1h + p2h)/2 < max_height)
  {
    #ifdef PRINT_INFO
    printf("  Rejected: Dash is too high.\n");
    #endif
```

【図159】

```
    return 0;
  }

  #ifdef PRINT_INFO
  printf(" Min height = %d\n", min_height, (p11 + p21)/2);
  #endif
  if ((p11 + p21)/2 > min_height)
  {
    #ifdef PRINT_INFO
    printf(" Rejected: Dash is too low.\n");
    #endif
    return 0;
  }

  // It should be a dash!
  return p1-1;
}


//////////////////////////////////////////////////////////////////////
// Check if there is a dash on the right side
int cd_right_dash_cut(
    unsigned line,
    struct Character **ch,
    int *hist)
{
    int h;
    int t;
    int w_min;
    int w1;
    int p1;
    int p11, p1h;
    int p2, p21, p2h;
    float m;
    int min_height, max_height;

    // The line
    // Points to the character we're on (Modified
    // to skip past newly inserted divisions)
    // The histogram (previously allocated)

    // Height of the character
    // The thresh crossing point
    // The minimum width for a dash
    // The break (or a left dash
    // The width of the left dash
    // Various points of the dash
    // (See lab log notebook)
    // The slope of the dash
    // The heights of the dashes

    // Compute minimum thickness
    h = (*ch)->bot - (*ch)->top;
    t = (int) (((float) h) * CD_MAX_DASH_WIDTH);

    // Compute minimum length of dash
    w_min = (int) (((float) h) * CD_MIN_WIDTH_CONST);

    // Compute width of dash
    p1 = (*ch)->right-1;
    while (((hist[p1] < t) && (p1 > (*ch)->left))
      p1--;
    w1 = (*ch)->right - p1;

  #ifdef PRINT_INFO
    printf("Right Side:\n");
    printf(" P1 = %d  W1 = %d\n", p1, w1);
  #endif

    // Reject if width is too short
    if ((w1 < w_min) || (w1 < MIN_ABS_WIDTH_TO_CONSIDER))
    {
      #ifdef PRINT_INFO
      printf(" Rejected: Width of dash is too short.\n");
      #endif
      return 0;
    }

    // Compute p1, p2. p1h, p2h, p11, p21    // Find positions of p1 and p2
    p1 = w1/4 + p1;
    for (p1h=(*ch)->top; p1h<=(*ch)->bot; p1h++) // Find p1h
      if ((p1 >= (*ch)->curve[p1h]) && (p1 < (*ch)->curve[p1h]))
        break;
    for (p11=(*ch)->top-1; p11 >= (*ch)->top; p11--)      // Find p11
      if ((p1 >= (*ch)->curve[p11]) && (p1 < (*ch)->curve[p11]))
        break;
    for (p2=p1+(*ch)->bot-1; p2 >= (*ch)->top; p2--) // Find p2h
      if ((p2 >= (*ch)->curve[p2h]) && (p2 < (*ch)->curve[p2h]))
        break;
    for (p21=(*ch)->bot-1; p21 >= (*ch)->top; p21--)      // Find p21
      if ((p2 >= (*ch)->curve[p21]) && (p2 < (*ch)->curve[p21]))
        break;
  #ifdef PRINT_INFO
    printf(" p1 = %d, p2 = %d\n", p1, p2);
    printf(" p1h, p11  = %d, %d\n", p1h, p11);
    printf(" p2h, p21  = %d, %d\n", p2h, p21);
  #endif

    // Check slope
    m = ((float) (p2h - p1h - p11)) / ((float) w1);
  #ifdef PRINT_INFO
    printf(" slope = %f\n", m);
  #endif
    if (m > CD_MAX_SLOPE_OF_DASH)
    {
      #ifdef PRINT_INFO
      printf(" Rejected: Dash slopes too much.\n");
      #endif
      return 0;
    }

    // Check height
    max_height = (int) (CD_TOO_HIGH * ((float) h) + (*ch)->top);
    min_height = (int) (CD_TOO_LOW * ((float) h) + (*ch)->top);

  #ifdef PRINT_INFO
    printf(" Max height = %d, top height = %d\n", max_height, (p1h + p2h)/2);
  #endif
    if ((p1h + p2h)/2 < max_height)
    {
      #ifdef PRINT_INFO
      printf(" Rejected: Dash is too high.\n");
      #endif
      return 0;
    }

  #ifdef PRINT_INFO
    printf(" Min height = %d, bot height = %d\n", min_height, (p11 + p21)/2);
  #endif
    if ((p11 + p21)/2 > min_height)
    {
      #ifdef PRINT_INFO
      printf(" Rejected: Dash is too low.\n");
      #endif
      return 0;
    }

    // It should be a dash!
    return p1-1;
}
```

【図160】

```
//////////////////////////////////////////////////////////////////////
// Check for the dash between a character, and break it if needed
void cd_chk(
  clangen line,          // The line
  struct CharList *clist,// Pointer to charlist
  struct Character *ch,  // Pointer to the character we're on (Modified
                         // to skip past newly inserted divisions)
  int *hist)             // The histogram (previously allocated)
{
  int p;

  // Reject if pixel width is too small.
  if (((*ch)->right - (*ch)->left) <= MIN_ABS_WIDTH_TO_CONSIDER)
    return;

#ifdef PRINT_INFO
  xm_clr_win();
  print_character(line, *ch, 10, 10);
  xm_flush();

  printf("\nCharacter info:\n");
  printf("   top/bot = (%d, %d)\n", (*ch)->top, (*ch)->bot);
  printf("   left/right = (%d %d)\n", (*ch)->left, (*ch)->right);
#endif

  // Compute the histogram
  i32_histogram(line, ch, hist);

  // Break left side?
  p = cd_left_dash_cut(line, ch, hist);
  if (p != 0)
  {
    i32_cut_character(line, clist, ch, p);  // Cut the character
#ifdef PRINT_INFO
    printf("Left dash cut\n");
    printf("\nCharacter info:\n");
    printf("   top/bot = (%d, %d)\n", (*ch)->top, (*ch)->bot);
    printf("   left/right = (%d %d)\n", (*ch)->left, (*ch)->right);
#endif
  }

  // Break right side?
  p = cd_right_dash_cut(line, ch, hist);
  if (p != 0)
  {
    i32_cut_character(line, clist, ch, p);  // Cut the character
#ifdef PRINT_INFO
    printf("Right dash cut\n");
#endif
  }
}

//
//.....
// MAIN ROUTINE
//.....
//
// This will cut the dashes of of characters
void cut_dash(clangen line,    // The clangen
  struct CharList* clist)      // The current charlist
{
  int *histogram;              // The area of the histogram
  struct Character *charon;    // The character we're processing

#ifdef PRINT_INFO
  int dummy;
#endif

  histogram = ivector(llite.sizeC);  // Allocate the memory

  charon = clist->first;             // Loop untill end of list
  while (charon != 0)
  {
    cd_chk(line, clist, &charon, histogram);
#ifdef PRINT_INFO
    printf("Enter 0 to continue? ");
    scanf("%d", &dummy);
#endif
    charon = charon->next;           // Get the next character
  }

  free_ivector(histogram);           // Deallocate the memory
}
```

```
//
//  Filename: level3m2.C
//  This file contains the routines to do level 3 segmentation (slightly
//  touching) using method 2 (histogram approach)
//
////////////////////////////////////////////////////////////////////////////
//


////////////////////////////////////////////////////////////////
// The includes
////////////////////////////////////////////////////////////////
#include"charlist.h"
#include"cimages.h"
#include"myalloc.h"


////////////////////////////////////////////////////////////////
//  The CONSTANTS
#define MIN_CUT_THRESHOLD 2
#define HISTORESIS_THRESHOLD 10


////////////////////////////////////////////////////////////////
// This routine will adjust left, right, curve1, curve2, top, bottom
// and pos to reposition a new character in the charlist, then it
// calls charlist_replace to reposition the new character
// Possible optimisation: Usually, this is called when a character is
//    split, so shrinkwrapping one side is 'in vain'
void l3_shrinkwrap_char(
  cimage& line,                 // The line
  struct Character **ch)        // Ptr to the character we're on
{
  int top, bot, left, right;    // The new left and right stuff.
  int r,c;
  int loc, loc_count, pos;      // pos = position, loc = pos * 8
  int *curve1, *curve2;         // Shorthand, set to (*ch)->curve1 and 2
  int flag;

  curve1 = (*ch)->curve1;
  curve2 = (*ch)->curve2;

  // Adjust the top of the character, if it needs to move down
  flag = 0;                     // top found when flag == 1;
  for (top = (*ch)->top; (top < (*ch)->bot) && (flag == 0); top++)
    for (c=curve1[top]; c < curve2[top]; c++)
      if (line.pixel[top][c] & 1)  // Stop at the first on pixel
      {
        flag = 1;
        break;
      }

  // Adjust the bottom of the character
  flag = 0;                     // bot found when flag == 1;
  for (bot = (*ch)->bot-1; (bot >= (*ch)->top) && (flag == 0); bot--)
    for (c=curve1[bot]; c < curve2[bot]; c++)
      if (line.pixel[bot][c] & 1) // Stop at the first on pixel
      {
        flag = 1;
        break;
      }
  bot++;                        // (Make bot exclusive)

  // Now adjust curve1 and curve2, updating left and right to 'shrink'
  // tightly around the border of the character (move the borders up to the
  // edge of the characters.
  loc = loc_count = 0;          // for position of the char (on the fly)
  for (r=top; r<bot; r++)
  {
    while (curve1[r] < curve2[r])       // Loop until they're equal
      if ((line.pixel[r][curve1[r]] & 1)) // If there is a non empty space
                                  // to the right of the left border
        break;                    // Then exit (left border not moved)
      else                        // There's an empty space to the right
        curve1[r] += 1;           // Slide the border over to meet it
    while (curve1[r] < curve2[r])       // Do the same with the right borde:
      if ((line.pixel[r][curve2[r]-1] & 1))
        break;
      else
        curve2[r] -= 1;

    if (curve1[r] != curve2[r])         // there is a pixel on this line...
    {
      loc += curve2[r] + curve1[r];
      loc_count++;
    }
  }
  if (loc_count != 0)
  {
    loc = loc * 4 / loc_count;  // Compute the position (used for .loc)
    pos = loc/8;                        // Compute the real position
  }

  // Find left and right
  left = curve1[top];
  right = curve2[top];
  for (r=top; r<bot; r++)
  {
    if (curve1[r] < left)
      left = curve1[r];
    if (curve2[r] > right)
      right = curve2[r];
  }

  if (loc == 0)
  {
    loc = 4 * (left+right);
    pos = (left+right)/2;
  }

  for(r=top; r<bot; r++)        // Fill in the blank lines with pos
    if (curve1[r] >= curve2[r])
      curve1[r] = curve2[r] = pos;

  // Copy all the new attributes in
  (*ch)->loc = loc;
  (*ch)->top = top;
  (*ch)->bot = bot;
  (*ch)->left = left;
  (*ch)->right = right;
}



////////////////////////////////////////////////////////////////
// This routine will cut **ch into two at point p.  clist is updated,
// and **ch returns so the *ch points to the right half of the character
void l3_cut_character(
    cimage& line,               // The line
    struct Charlist *clist,     // The character list
    struct Character **ch,      // The character to be cut
    int p)                      // Where to cut the character
{
```

```
  struct Character *newchar;    // The new character
  int i;

  newchar = &(clist->chr[clist->size]);  // Set newchar to point to the end
                                         // of the charlist (the next blank character)

  for (i=0; i<clist->vector_height; i++)  // Loop through the char (up-down)
  {
    newchar->curve1[i] = (*ch)->curve2[i];      // Break the character
    newchar->curve2[i] = (*ch)->curve2[i] = p;
    if (newchar->curve1[i] < p)  // Make sure curves of new char are >= p
      newchar->curve1[i] = p;
    if (newchar->curve2[i] < p)
      newchar->curve2[i] = p;
    if ((*ch)->curve1[i] > p)    // Make sure curves of old char are <= p
      (*ch)->curve1[i] = p;
    if ((*ch)->curve2[i] > p)
      (*ch)->curve1[i] = p;

  }
  newchar->top = (*ch)->top;
  newchar->bot = (*ch)->bot;
  newchar->left = p;
  newchar->right = (*ch)->right;
  (*ch)->right = p;

  l32_shrinkwrap_char(line, &newchar);
  charlist_place_new_char(clist);       // Place newchar into the charlist
  l32_shrinkwrap_char(line, ch);
  charlist_replace_char(clist, *ch);

  *ch = newchar;                // Adjust ch to the new character
}


////////////////////////////////////////////////////////////////////////
// Compute the histogram of a character
void l32_histogram(                 // Compute the histogram
  cimage4 line,                     // The line
  struct Character **ch,            // Pointer to the character we're on (Modified
                                    // to skip past newly inserted divisions)
  int *hist)                        // The histogram array (output)
{
  int r,c;
  // Zero the histogram
  for (c=(*ch)->left; c < (*ch)->right; c++)
    hist[c] = 0;
  for (r=(*ch)->top; r < (*ch)->bot; r++)       // Compute histogram
    for (c=(*ch)->curve1[r]; c < (*ch)->curve2[r]; c++)
      if (line.pixel[r][c] & 1)
        hist[c]++;
}


////////////////////////////////////////////////////////////////////////
// Check if a character need to be broken, and break it if so
void l32_insert_breaks(
  cimage4 line,                     // The line
  struct Charlist *clist,           // Pointer to charlist
  struct Character **ch,            // Pointer to the character we're on (Modified
                                    // to skip past newly inserted divisions)
  int *hist,                        // The histogram (previously allocated)
  int min_cut_threshold,            // The minimum value to make a cut
  int historesis_threshold)         // Historesis threshold ( >min_cut_threshold)
{
  int p;                            // The cursor
  int p_end;                        // The end of the cursor
  int min, min_at;                  // Where the minimum value is
  int l,r;                          // Left and right boundries of where to search
  int c;

  // Compute the histogram
  l32_histogram(line, ch, hist);

  p_end = (*ch)->right;
  p = (*ch)->left;                  // Start at left side
  while (p < p_end)                 // Exit when we've scaned the entire histogram
  {
    while (++p < p_end)             // Looking for top-bottom crossing of threshold
      if ((hist[p-1] > historesis_threshold) &&
          (hist[p] <= historesis_threshold))
        break;
    l = p;                          // This is the left side of the area to search

    while (++p < p_end)             // Looking for bottom-top crossing of threshold
      if ((hist[p-1] <= historesis_threshold) &&
          (hist[p] > historesis_threshold))
        break;
    r = p;                          // Remember right edge;

    if (p < p_end)                  // If we haven't gone over the edge
    {
      // Search for min in l_to_r range
      min = hist[l];
      min_at = l;
      for (c=l+1; c<r; c++)
        if (hist[c] < min)
        {
          min = hist[c];
          min_at = c;
        }

      // If the min is small enough, cut there
      if (min <= min_cut_threshold)
      {
        l32_cut_character(line, clist, ch, min_at);  // Cut the character
                                    // Note: ch is changed to the right char,
                                    // which shouldn't mess things up
        printf("Level 3 made a cut.\n");
      }
    }
  }
}


//
//*****************************************************************
// MAIN ROUTINE
//*****************************************************************
//
// This will do histogram touching on the characters for level 2 method 3

void level3m2(cimage4 line,             // The cimage
    struct Charlist* clist)     // The current charlist
{
  int *histogram;               // The area of the histogram
  struct Character *charon;     // The character we're processing

  histogram = ivector(line.sizeC);      // Allocate the memory

  charon = clist->first;        //
  while (charon != 0)           // Loop until end of list
  {
    l32_insert_breaks(line, clist, &charon, histogram, MIN_CUT_THRESHOLD,
```

【図164】

```
//
// Filename: linprocess.c
// This routine contains stuff to do character segmentation
//
//

#include"charlist.h"
#include"cimages.h"
#include"xs.h"
#include"mymalloc.h"
#include"autoinput.h"


//#define PASS_TO_LEVEL_3      // Defines when runs level 2
//#define PASS_TO_LEVEL_2      // Define if to run level 3
#define PASS_TO_LEVEL_4 // Call the level 4 routines
//#define COMBINING       // Call the combining routines
//#define BASE_CHARS      // Call the routines to cut dashed characters
//#define PRINT_FINAL_CHARLIST

// Declare external routines
void extract_zero_vpp_list(cimage& line, struct NonZeroVppList *list);
void do_splitting(cimage& line, struct Charlist* clist, int start_col,
                  int stop_col);
void do_slightly_touching(cimage& line, struct Charlist* clist, int start_col,
                  int stop_col);
void charlist_combine(cimage& line, struct Charlist*);

// Declare forward referenced internal routines
void print_last_charlist(cimage& line, struct Character *clist, int x, int y);

// Define global variables
int cursor_x, cursor_y; // Screen coordinates

extern int WINDOW_SIZE_X;
extern int WINDOW_SIZE_Y;

extern int Logic_type;          // 0 for none, 1 for OR, 2 for AND-OR
cimage Process_line;            // The line being segmented (so not reallocated)

// Optimal line routines
extern void cline_OR_1(cimage& oldline, cimage& newline);
extern void cline_OR_2(cimage& oldline, cimage& newline);
extern void cline_AND_1?_OR_1(cimage& oldline, cimage& newline);
extern void cline_AND_1?_OR_1(cimage& oldline, cimage& newline);

// The compression logic variables (constants)
extern int RESOLUTION_LINE_X_AND;
extern int RESOLUTION_LINE_C_AND;
extern int RESOLUTION_LINE_X_OR;
extern int RESOLUTION_LINE_C_OR;
extern int RESOLUTION_LINE_X_SCALED;
extern int RESOLUTION_LINE_C_SCALED;


//////////////////////////////////////////////////////////////////////////////
// Misc utility function // Returns minimum of two values
int min(
    int a,
    int b)
{
  if (a>b)
    return b;
  return a;
}

//////////////////////////////////////////////////////////////////////////////
// Print a character on the screen
void print_character(
    cimage& line,           // The image of the line
    struct Character *chr,  // A pointer to the character
    int x,                  // The position
    int y)
{
  int r,c;
  for (r=chr->top; r < chr->bot; r++)
    for (c=chr->curve1[r]; c < chr->curve2[r]; c++)
      if (line.pixel[r][c] & 1)
        xs_plot(x + c - chr->left, y + r);

  // Plot the outlined border
  for (r=chr->top; r<=chr->bot; r++)
  {
    xs_plot(x + chr->curve1[r] - 2 - chr->left, y + r);
    xs_plot(x + chr->curve2[r] + 2 - chr->left, y + r);
  }

  // Plot the underline
  for (c=chr->left; c < chr->right; c++)
    xs_plot(x + c - chr->left, y + line.size5-5);

  // Plot the tag in the underline
  xs_plot(x + chr->left + chr->loc[0], y + line.size4 + 4);
  xs_plot(x + chr->left + chr->loc[0], y + line.size4 + 6);
}

//////////////////////////////////////////////////////////////////////////////
// Print the last part of a characterlist on the screen (starting at chr)
void print_last_charlist(
    cimage& line,           // The image of the line
    struct Character *chr,  // The position
    int x,
    int y)
{
  int x_start;              // The x starting position (for wrap around)

  x_start = x;              // Remember starting position

  while(chr != 0)           // Loop through entire list
  {

    if (x - chr->right - chr->left > WINDOW_SIZE_X - 20) // Wrap?
    {
      x = x_start;
      y += line.size4 + 15;
    }
    print_character(line, chr, x, y);
    x += chr->right - chr->left + 10;
    chr = chr->next;            // Get next character
  }
}

//////////////////////////////////////////////////////////////////////////////
// Print a characterlist on the screen
void print_charlist(
    cimage& line,           // The image of the line
    struct Charlist *clist,  // A pointer to the character list
```

[図165]

```
    int x,                      // The position
    int y)
{
  struct Character *chr;

  chr = clist->first;           // Point to first character
  print_last_charlist(line, chr, x, y);
}


////////////////////////////////////////////////////////////////////
// Print a nonzerovpplist on the screen
void print_nzvpplist(
    cimage& line,
    struct NonZeroVppList *nzlist,
    int x,                      // The position
    int y)
{
  int x_start;                  // The x starting position (for wrap around)
  int i,r,c;

  x_start = x;                  // Remember starting position
  for (i=0; i<nzlist->size; i+=2)
  {
    for (r=0; r<line.sizeR; r++)
      for (c=nzlist->list[i]; c<=nzlist->list[i+1]; c++)
        if (line.pixel[r][c] & 1)
          xs_plotxy(x + c, y + r);
    for (r=0; r<line.sizeR; r++)        // Plot double line
    {
      xs_plotxy(x+nzlist->list[i]-1, y+r);
      xs_plotxy(x+nzlist->list[i+1], y+r);
    }

    x += 5;
    if (x > WINDOW_SIZE_X - 50) //  Wrap?
    {
      x = x_start;
      y += line.sizeR + 15;
    }
  }
}


////////////////////////////////////////////////////////////////////
// This will look at a segment, after segmented from level 1 segmentation
// and decide of it needs to go to level 2
int decide_level12(     // Returns 1 for go to level 2, 0 otherwise
    cimage& line,       // The image of the line to segment
    int start_col,      // The first column of where the character is (incl)
    int stop_col)       // The last column of this chere\
{
  // Delete following. Removes warnings during compilation
  int a = start_col + stop_col + line.size();

  return 1;
}


//*****************************************************************
//*****************************************************************
//**********                    MAIN                     **********
//*****************************************************************
//*****************************************************************
void process_line(
    cimage& input_line,         // The image of the line to segment
    struct Charlist *clist)     // Output: The characterlist.  Should
                                // be initialized outside this routine
{
  cimage *line_to_process;      // The line to process
  struct NonZeroVppList *nzlist;
  int i;
  int newr, newc, j,m,n, stop_m, stop_n;
  register char new_pixel_value;

//************************** NO LOGIC ****************************
  line_to_process = &(input_line);              // The default

//************************** THE OR LOGIC ************************
  if (Logic_type == 1)
  {
    if (((RESOLUTION_3LINE_R_OR == 2) && (RESOLUTION_3LINE_C_OR == 1)) ||
        ((RESOLUTION_3LINE_R_OR == 3) && (RESOLUTION_3LINE_C_OR == 1)))
    {
      if ((RESOLUTION_3LINE_R_OR == 2) && (RESOLUTION_3LINE_C_OR == 1))
      {
        printf("**** Using optimal OR 3x1 routine\n");
        tline_OR_31(input_line, Process_line);
      }
      if ((RESOLUTION_3LINE_R_OR == 3) && (RESOLUTION_3LINE_C_OR == 1))
      {
        printf("**** Using optimal OR 3x1 routine\n");
        tline_OR_31(input_line, Process_line);
      }
    } else {            // Do standard
      newr = (input_line.sizeR-1)/RESOLUTION_3LINE_R_OR+1;
      newc = (input_line.sizeC-1)/RESOLUTION_3LINE_C_OR+1;
      Process_line.reallocate(newr,newc);

      for (i=0; i<newr; i++)
        for (j=0; j<newc; j++)
        {
          new_pixel_value = 0;
          stop_m = imin((i+1)*RESOLUTION_3LINE_R_OR, input_line.sizeR);
          stop_n = imin((j+1)*RESOLUTION_3LINE_C_OR, input_line.sizeC);
          for (m=i*RESOLUTION_3LINE_R_OR; m < stop_m; m++)
            for (n=j*RESOLUTION_3LINE_C_OR; n< stop_n; n++)
              new_pixel_value = new_pixel_value | input_line.pixel[m][n];
          Process_line.pixel[i][j] = new_pixel_value;
          }

    }
    line_to_process = &(Process_line);
  }

//************************** THE AND OR LOGIC ********************
  if (Logic_type == 2)
  {
    if (((RESOLUTION_3LINE_R_AND == 1) && (RESOLUTION_3LINE_C_AND == 2) &&
         (RESOLUTION_3LINE_R_OR == 2) && (RESOLUTION_3LINE_C_OR == 1)) ||
        ((RESOLUTION_3LINE_R_AND == 3) && (RESOLUTION_3LINE_C_AND == 2) &&
         (RESOLUTION_3LINE_R_OR == 3) && (RESOLUTION_3LINE_C_OR == 1)))
    {
      if ((RESOLUTION_3LINE_R_AND == 1) && (RESOLUTION_3LINE_C_AND == 2) &&
          (RESOLUTION_3LINE_R_OR == 2) && (RESOLUTION_3LINE_C_OR == 1))
      {
        printf("**** Using optimal AND 1x2 OR 3x1 routine\n");
        tline_AND_12_OR_31(input_line, Process_line);
      }
      if ((RESOLUTION_3LINE_R_AND == 1) && (RESOLUTION_3LINE_C_AND == 2) &&
          (RESOLUTION_3LINE_R_OR == 3) && (RESOLUTION_3LINE_C_OR == 1))
      {
        printf("**** Using optimal AND 1x2 OR 3x1 routine\n");
        tline_AND_12_OR_31(input_line, Process_line);
      }
```

[図166]

```
    } else {            // Do standard
      // The AND
      newr = (input_line.sizeR-1)/RESOLUTION_3LINE_R_AND+1;
      newc = (input_line.sizeC-1)/RESOLUTION_3LINE_C_AND+1;
      Process_line.reallocate(newr,newc);

      for (i=0; i<newr; i++)
        for (j=0; j<newc; j++)
        {
          new_pixel_value = 1;
          stop_m = lmin((i+1)*RESOLUTION_3LINE_R_AND, input_line.sizeR);
          stop_n = lmin((j+1)*RESOLUTION_3LINE_C_AND, input_line.sizeC);
          for (m=i*RESOLUTION_3LINE_R_AND; m < stop_m; m++)
            for (n=j*RESOLUTION_3LINE_C_AND; n< stop_n; n++)
              new_pixel_value = new_pixel_value & input_line.pixel[m][n];
          Process_line.pixel[i][j] = new_pixel_value;
          }

      // The OR
      newr = (Process_line.sizeR-1)/RESOLUTION_3LINE_R_OR+1;
      newc = (Process_line.sizeC-1)/RESOLUTION_3LINE_C_OR+1;

      for (i=0; i<newr; i++)
        for (j=0; j<newc; j++)
        {
          new_pixel_value = 0;
          stop_m = lmin((i+1)*RESOLUTION_3LINE_R_OR, Process_line.sizeR);
          stop_n = lmin((j+1)*RESOLUTION_3LINE_C_OR, Process_line.sizeC);
          for (m=i*RESOLUTION_3LINE_R_OR; m < stop_m; m++)
            for (n=j*RESOLUTION_3LINE_C_OR; n< stop_n; n++)
              new_pixel_value = new_pixel_value | Process_line.pixel[m][n];
          Process_line.pixel[i][j] = new_pixel_value;
        }
      Process_line.reallocate(newr,newc);       // Readjust size afterwards
    }
    line_to_process = &(Process_line);
  }

//************************** SCALED LOGIC ************************
  if (Logic_type == 3)
  {
    newr = (input_line.sizeR-1)/RESOLUTION_3LINE_R_SCALED+1;
    newc = (input_line.sizeC-1)/RESOLUTION_3LINE_C_SCALED+1;
    Process_line.reallocate(newr,newc);
    for (i=0; i<newr; i++)
      for (j=0; j<newc; j++)
        Process_line.pixel[i][j] =
              input_line.pixel[i*RESOLUTION_3LINE_R_SCALED]
                              [j*RESOLUTION_3LINE_C_SCALED];
    line_to_process = &(Process_line);
  }

//******************** END OF LOGIC ROUTINES ************************

#ifdef VISIBLE
  xs_clr_win();
  (*line_to_process).plot(10,10);
  xs_flush();
#endif


  struct NonZeroVppList masternzlist;   // *** DELETE ***/
  Zerovpp_initialize(&masternzlist, 200);
  nzlist = &masternzlist;       // ***** ADD THIS HIGHER!!! *****/

#ifdef VISIBLE
  xs_clr_win();
    cursor_X = 10;          // The screen coordinates
    cursor_Y = 10;

  //  (*line_to_process).plotbox(cursor_X, cursor_Y);
  //  xs_flush();
    cursor_Y += (*line_to_process).sizeR + 10;
#endif

    ********** Insert breaks caused by zero values of VPP
    nzlist->size = 0;               // Zero the list
    extract_zero_vpp_list((*line_to_process), nzlist);

#ifdef PASS_TO_LEVEL_2
    ********** Insert outlined breaks
    Charlist_zero(clist);                   // Empty the character_list
    for (i=0; i<nzlist->size; i+=2)         // Loop through each character
    {
      if (decide_level2((*line_to_process), nzlist->list[i], nzlist->list[i+1]))
      {
        do_outlining((*line_to_process), clist, nzlist->list[i], nzlist->list[i+1]);
        nzlist->list[i] = -1; nzlist->list[i+1] = -1;
                        // Indicate that this is in a charlist
      }
    }

    ********** Combine sections that should be combined (in the charlist)
    #ifdef COMBINING
    charlist_combine((*line_to_process), clist);
    #endif
#else
    struct Character *nc;
    // Simply convert the nzlist to the charlist
    Charlist_zero(clist);                   // Empty the character_list
    for (i=0; i<nzlist->size; i+=2)
    {
      nc = &(clist->chr[clist->size]); // Set new_char to where it's going
      nc->top = 0;
      nc->bot = (*line_to_process).sizeR;
      nc->left = nzlist->list[i];
      nc->right = nzlist->list[i+1];
      for (j=0; j<(*line_to_process).sizeR; j++)
      {
        nc->curve1[j] = nzlist->list[i];
        nc->curve2[j] = nzlist->list[i+1];
      }
      Charlist_place_new_char(clist);
    }
#endif


#ifdef PASS_TO_LEVEL_3
    extern void level3m2(cimage& line, struct Charlist* clist);

    ********** Insert slightly touching breaks
    level3m2((*line_to_process), clist);
#endif


#ifdef PASS_TO_LEVEL_4
    extern void level4(cimage& line, struct Charlist *clist);

    level4((*line_to_process), clist);
#endif


#ifdef DASH_CHARS
```

[図168]

```
//
// Filename: multilevel.C
// This routine contains stuff to do layer 4 segmentation
//
////////////////////////////////////////////////////////////////
//

extern "C" printbreaks(struct Breaklist blist);
extern "C" printbreaks_ptr(struct Breaklist *blist);
extern "C" int recognized(int **a, int size_r, int size_c);

#include"images.h"
#include"xs.h"
#include"breaklist.H"
#include"mymalloc.h"
#include"charlist.h"
#include"cimages.h"
#include"autoinput.h"

void insert_outline_breaks(gimage& image, Breaklist* blist,
                           int bnum1, int bnum2);
void print_last_charlist(cimage& line, struct Character *clist, int x, int y);
void 132_shrinkwrap_char(cimage& line, struct Character **ch);

#define MIN_CHAR_WIDTH 6          // The minimum width to be cut
#define MAX_CHAR_WIDTH 51         // The Maximum character width (will
                                  // automatically break if width is > this)
                                  // (These are approximately in pixels)

#define MAX_CHARS_PER_CHUNK 100 // Defines maximum number of character in a
                                  // piece sent to multilevel

#define MAX_BREAKS 500            // Maximum number of breaks in an image
#define MAX_BREAKV 500            // Maximum number of curvy breaks in image
#define MAX_NUM_SEGMENTS 500
#define MAX_VPP_SIZE 500          // Maximum image size for VPP array
#define SEGMENT_VPP_FRACTION 0.5 // Fraction of height of VPP to segment

// Define global variables
extern int WINDOW_SIZE_X;
extern int WINDOW_SIZE_Y;


//************** WARNING: CHANGE THESE BACK TO CONSTANTS!!!!!! ************
#define LR_IGNORE_HEIGHT 0.25   // Minimum valley height to clip left and
                                  // Edges after taking rotated projection (so
                                  // minimum values on ends will be ignored

int AVPP_compute_point(int r, int c, double theta); // Define routine
void print_image_with_breaks(gimage& image, Breaklist* blist, int break1,
        int break2, int x, int y);

// Define temporary global variables
int mcursor_X = 10;     // The screen coordinates
int mcursor_Y = 10;
int cursor1_X = 10;     // The screen coordinates
int cursor2_Y = 10;
int cursor3_X = 10;     // The screen coordinates
int cursor3_Y = 10;
int cursor_col_left = 0;
int cursor_col_right;
int cursor_top_cline;

//----------------------------------------------------------------
////////////////////////////////////////////////////////////////


////////       Find Projection at a rotated angle       ////////
////////////////////////////////////////////////////////////////
// Note: This routine is extremely unoptimized for speed and space!
// Input: image, theta,
//      blist=the breaklist, bnum is the break on the left side to start
//            searching from
//      allocated_a = highest index allocated for a
// Ouput: a = VPP, max_a = highest VPP index value (output),
//      index_0a = what point a[0] corresponds to.
#define F_P_STORAGE 10000  // Memory allocated for projection
#define F_P_ZERO 15000     // Where the zero is
void find_projection(gimage& image,     // The input image
        double theta,           // Angle of the rotated axis (-pi/2..pi/2)
        struct Breaklist* blist, // The breaklist for the image
        int bnum1,              // break # of left side for computing profile
        int bnum2,              // break # of right side for computing profile
        int c,                  // Right point to search for
        int allocated_a,        // Max sized of a[] allocated
        int a[],                // Output: The profile
        int& max_a,             // Output: The highest index of a used
        int& index_0a)          // Output: What point a[0] corresponds to
{
  int store[F_P_STORAGE]; // The profile
  int start_store, stop_store; // The highest values used in store
  int i;                  // Misc. Vars
  int r,c;
  int p;

  int *leftbreak = new int[image.sizeR];  // The start and stop locations
  int *rightbreak = new int[image.sizeR];

  // Initialize for computing the Angular VPP
  for (r=0; r < F_P_STORAGE; r++)  store[r] = 0;        // Zero a matrix
  start_store = F_P_STORAGE;    // Define the bounds to zero
  stop_store = 0;
  Breaklist_return_curves(blist, bnum1, bnum2, leftbreak, rightbreak);

  // Compute the Angular VPP
  for (r=0; r < image.sizeR; r++)       // Loop through image
    for (c=leftbreak[r]; c<rightbreak[r]; c++)  // Loop through image
    {
      if (c >= C-(image.sizeR-r)*tan(theta))  // If it's outside the line
        {
          break;
        }               // Exit from looping through c
      if (image.pixel[r][c] != 0)
      {
        p = AVPP_compute_point(r, c, theta)+F_P_ZERO;   // Calculate point
        if ((p < 0) || (p > F_P_STORAGE))  // Check range
        {
          cerr << "Error in find_projection(): F_P_STORAGE too small!\n";
          exit(1);
        }
        store[p]++;             // Increment count
        if (p < start_store)  start_store = p;  // Update max and min range
        if (p > stop_store)   stop_store = p;
      }
    }

  // Deallocate memory used
  delete [image.sizeR] leftbreak;
  delete [image.sizeR] rightbreak;

  // Copy store to a with information
  if ((start_store == F_P_STORAGE) && (stop_store == 0))  // If no projection
  {
    max_a = 0;  // Return size of 0
```

```
    index_0a = 0;  // Set just in case
    return;      // Exit
  }

  max_a = stop_store - start_store + 1;
  if (max_a > allocated_a)
  {
    cerr << "Error in find_projection: ";
    cerr << "Not enough memory allocated for matrix a()\n";
    exit(1);
  }
  index_0a = start_store-F_P_ZERO;
  for (i=start_store; i<= stop_store; i++)       // Copy store to a
    a[i-start_store] = store[i];
}


// Convert a double to an int by clipping down:
// i.e. -0.5 -> -1, 1.7 -> 1
int clipdouble(double i)
{
  if (i >= 0)
    return (int) i;
  else
    return (int) i-1;
}


// Computes what colluma of the rotated histogram point (r,c) falls
// in when the axis is rotated theta degrees
int AVPP_compute_point(int r, int c, double theta)
{
  double dr, dc;          // Row and Column
  double dl;              // Misc. var (length of diagonal to r,c)
  double dp;              // Output variable
  double alpha;           // Misc. var (angle of (1,0) to (r,c))

  dr = (double) r+.5;   // Convert to double
  dc = (double) c+.5;   //   (the +.5 gets the center of the box)
  alpha = atan(dr/dc);
  dl = sqrt(dr*dr + dc*dc);
  dp = dl * cos(alpha-theta);
  return clipdouble(dp);
}


//*************************************************************
////////////////////////////////////////////////////////////////
////////        The product routine to do thinning      ////////
////////////////////////////////////////////////////////////////
// Overlap: C = A overlapped with B (times)  (must be same dimensions!)
void product(dimage& C, dimage& A, dimage& B)
{
  int i,j;

  C.fill(0, A.sizeR, A.sizeC);
  for (i=0; i<A.sizeR; i++)
    for (j=0; j<A.sizeC; j++)
      C.pixel[i][j] = A.pixel[i][j] * B.pixel[i][j] / 255.0;
}


////////////////////////////////////////////////////////////////
////////          This will do thinning             ////////
////////////////////////////////////////////////////////////////
void thin(dimage& in_image, double sigma, dimage& out_temp)
{
  dimage temp, t1, t2, t3;
  int tempsize;
  int i,j;

  ////////// Make the gaussian dimage ///////////
  if (sigma < 1.0) sigma = 1.0;          // Get dimage size
  tempsize = (int) sigma * 6;
  if ((tempsize % 2) == 0)
    tempsize++;
  temp.makegauss(sigma, tempsize);       // Make dimage

  ////////// Blur degraded image by sigma //////////////
  t1.convolve(in_image, temp);  // t1 = blurred image

  t2.computeddt(t1);             // t2 = derivative of blurred image
  t2.scale(0,255);

  t3.fill(0,t2.sizeR, t2.sizeC); // t3 = invered derivative of blurred image
  for (i=0; i<t2.sizeR; i++)
    for (j=0; j<t2.sizeC; j++)
      t3.pixel[i][j] = 255.0 - t2.pixel[i][j];

  product(out_temp, t3, t1);     // t4 = blurred * derivative (product)
}


////////////////////////////////////////////////////////////////
/////          Plot one line through a point             //////
////////////////////////////////////////////////////////////////
// (This plots point for point in the y direction (so it is to be
//  used with more vertical lines.)
void plot_line_at_angle(
  int r_point,          // The x coord of the point
  int c_point,          // The y coord of the point
  double theta,         // The angle
  int y1,               // The y range to print
  int y2)
{
  double m, b, xd;
  int x,y;


  // Note: m and b are for x = my + b !!!!!!
  m = tan(theta);                // Compute the slope
  b =c_point - m*r_point;        // Compute intercept

  for (y=y1; y<y2; y++)
  {
    xd = m * (double) y + b;
    x = (int) xd;
    xs_plotxy(x,y);
  }
}


//*************************************************************
//********        The Segmentation Routines          ********
//*************************************************************
////////////////////////////////////////////////////////////////
////////      Insert breaks where has zero VPP          ////////
////////////////////////////////////////////////////////////////
void insert_zeroVPP_breaks(
  breaklist* blist,             // The breaklist to insert breaks in
  int *proj,                    // The VPP (previously calculated)
  int proj_size,                // The max value of the VPP
  int proj_zero)                // What value column proj[0] corresponds to
{
```

[図170]

```
  int i;                        // The cursor in the projection
  int dummy;

  for (i=1; i<proj_size; i++)   // For each boarder between points i-1 & i
  {
    if ((proj[i-1] != 0) && (proj[i] == 0)) // If right boarder
      dummy = breaklist_add_break(blist, (i+proj_zero)*blist->bvector_height, BREAK_SRC
PACE_START, 0);
    if ((proj[i-1] == 0) && (proj[i] != 0)) // If left boarder
      dummy = breaklist_add_break(blist, (i+proj_zero)*blist->bvector_height, BREAK_SRC
PACE_STOP, 0);
  }
}



////////////////////////////////////////////////////////////////////////
//////        Find minimum value in a vector over a specified       //////
////////////////////////////////////////////////////////////////////////
void find_minimum_in_vector(
  int a[],                      // The input vector
  int r1,                       // The range to search over (inclusive)
  int r2,
  int& min,                     // Output: minimum value
  int& min_at)                  // Output: It's index
{
  int i;
  min = a[r1];                  // Set first value as default
  min_at = r1;
  for (i=r1+1; i<=r2; i++)
    if (a[i] < min)
    {
      min = a[i];
      min_at = i;
    }
}


////////////////////////////////////////////////////////////////////////
//////      Find minimum in a valley for a rotated profile          //////
////////////////////////////////////////////////////////////////////////
void find_min_valley_at_theta(
  gimage& image,                // The image
  struct breaklist* blist,      // The breaklist for the image
  int bnum1,                    // The range of breaks to search through
  int bnum2,
  int C,                        // Where to break on the right
  double theta,                 // The angle of rotation to compute
  int& min,                     // The minimum value (-1 if no good ones)
  int& min_at)                  // Where the minimum is located at (-1 = bad)
{
  int r1, r2;                   // Range to look for
  int *a, allocated_a;          // The rotated projection
  int max_a, index_0a;          // Info on the projection
  int max_value_a;              // The maximum value of the projection
  int i;

  // Allocate memory for a[]
  allocated_a = image.sizeC*image.sizeX + 1; // This should be enough
  a = ivector(allocated_a);

  // Find the rotated projection
  find_projection(image, theta, blist, bnum1, bnum2, C,
                  allocated_a, a, max_a, index_0a);

  // Find maximum projection
  max_value_a = a[0];
  for (i=1; i<max_a; i++)
    if (a[i] > max_value_a)
      max_value_a = a[i];

  // Choose range to ignore points before first peaks on left and right
  r1 = 0;
  r2 = max_a-1;

  // Find left peak (while slope is still rising), and above LR_IGNORE_HEIGHT
  while ((a[r1] < a[r1+1]) || (a[r1] < a[r1+2]) || (a[r1] < a[r1+3]) ||
         a[r1] < LR_IGNORE_HEIGHT*max_value_a)
    if (++r1 >= max_a - 3)          // If got to left edge
    {
      min = min_at = -1;            // Return no minimum found
      free_ivector(a);
      return;
    }

  // Find right peak (while slope is rising (from right to left))
  while ((a[r2] < a[r2-1]) || (a[r2] < a[r2-2]) || (a[r2] < a[r2-3]) &&
         a[r2] < LR_IGNORE_HEIGHT*max_value_a)
  {
    r2--;
    if ((r2 <= r1) || (r2 <=3))     // If reached the left point
    {
      min = min_at = -1;            // Return no minimum found
      free_ivector(a);
      return;
    }
  }

  find_minimum_in_vector(a, r1, r2, min, min_at);       // Find the minimum
  min_at += index_0a;           // Add the offset from the projection

  free_ivector(a);
}


////////////////////////////////////////////////////////////////////////
// TEMPORARY PRINTING ROUTINE: Prints find_min_valley_at_theta() info
////////////////////////////////////////////////////////////////////////
// **** DELETE ME EVENTUALLY ****  (the whole routine!)
void find_min_valley_at_theta_print(
  gimage& image,                // The image
  struct breaklist* blist,      // The breaklist for the image
  int bnum1,                    // The range of breaks to search through
  int bnum2,
  int C,                        // Where to break on the right
  double theta,                 // The angle of rotation to compute
  int& min,                     // The minimum value (-1 if no good ones)
  int& min_at)                  // Where the minimum is located at (-1 = bad)
                                // (Note: If theta = 0, this is the minimum!
{
  int r1, r2;                   // Range to look for
  int *a, allocated_a;          // The rotated projection
  int max_a, index_0a;          // Info on the projection
  int max_value_a;              // The maximum value of the projection
  int i, j;
  char string[200];

  // Allocate memory for a[]
  allocated_a = image.sizeC*image.sizeX + 1; // This should be enough
  a = ivector(allocated_a);

  // Find the rotated projection
  find_projection(image, theta, blist, bnum1, bnum2, C,
```

```
                allocated_a, a, max_a, index_0a);

// Find maximum projection
max_value_a = a[0];
for (i=1; i<max_a; i++)
  if (a[i] > max_value_a)
    max_value_a = a[i];

// Choose range to ignore points before first peaks on left and right
r1 = 0;
r2 = max_a-1;

if (theta != 0 )
{
  // Find left peak (while slope is still rising), and above LR_IGNORE_HEIGHT
  while ((a[r1] < a[r1+1]) || (a[r1] < a[r1+2]) || (a[r1] < a[r1+3]) ||
         a[r1] < LR_IGNORE_HEIGHT*max_value_a)
    if ((++r1) >= max_a - 3)            // If got to left edge
    {
      min = min_at = -1;                // Return no minimum found
      free_ivector(a);
      return;
    }

 // Find right peak (while slope is rising (from right to left))
  while ((a[r2] < a[r2-1]) || (a[r2] < a[r2-2]) || (a[r2] < a[r2-3]) &&
         a[r2] < LR_IGNORE_HEIGHT*max_value_a)
  {
    r2--;
    if ((r2 <= r1) || (r2 <=3))                 // If reached the left point
    {
      min = min_at = -1;                // Return no minimum found
      free_ivector(a);
      return;
    }
  }

  find_minimum_in_vector(a, r1, r2, min, min_at);      // Find the minimum
}


// ****** The above is mostly copied from void find_min_valley_at_theta ****
// ****** The following section is VERY VERY poorly written!!!!!!!!

for (i=cursor_col_left+1; i<cursor_col_right-1; i++)  // Draw line above
  xs_plotxy(i, cursor2_Y);
cursor2_Y += 8;

// Plot the image
print_image_with_breaks(image, blist, bnum1, bnum2, cursor2_X, cursor2_Y);

// Find position of the C line and plot it
struct Break b1;                // Find initial offset
Breaklist_return_break(blist, bnum1, &b1);    // Get first break
int Cx = cursor2_X - b1.loc/image.sizeR + C + 2;
int Cy = cursor2_Y + image.sizeR + 2;
plot_line_at_angle(Cy, Cx, theta, cursor2_Y - 4, Cy + 5); // Right line

cursor2_Y -= image.sizeR + 5.4;

// Plot the histogram
int tttemp = simple_plotmag_hist(a, max_a, cursor2_X, cursor2_Y);
sprintf(string, "%5.1f", theta/3.14159*180.0);
xs_string(cursor2_X + max_a + 5, cursor2_Y + tttemp-8, string);
if (theta != 0)                 // Plot dotted line
{
  int ttt = (int) (LR_IGNORE_HEIGHT * (double) max_value_a);

  for (i=cursor2_X + 5; i< cursor2_X + max_a + 4; i++)
    if ((i%2 == 0)
      xs_plotxy(i, cursor2_Y + tttemp - ttt );
}
cursor2_Y += tttemp + 2;

// Plot arrow information at bottom
j = cursor2_Y;
r1 += cursor2_X; r2 += cursor2_X;             // NOTE!!!

if (theta != 0)                 // Plot bar
{
  for (i=r1; i<=r2; i++)
    xs_plotxy(i, j+5);
  xs_plotxy(r1, j+6); xs_plotxy(r2, j+6);     // Plot ends of bar
  xs_plotxy(r1, j+7); xs_plotxy(r2, j+7);
  xs_plotxy(r1, j+4); xs_plotxy(r2, j+4);
  xs_plotxy(r1, j+3); xs_plotxy(r2, j+3);
}

// Draw arrow
if (theta == 0)
  min_at -= b1.loc/image.sizeR + 1;
i = min_at + cursor2_X;
j = cursor2_Y-1;
xs_plotxy(i,j);
xs_plotxy(i-1,j+1); xs_plotxy(i,j+1);  xs_plotxy(i+1,j+1);
xs_plotxy(i-2,j+2); xs_plotxy(i-1,j+2); xs_plotxy(i,j+2);
   xs_plotxy(i+1,j+2); xs_plotxy(i+2,j+2);
xs_plotxy(i, j+3); xs_plotxy(i, j+4); xs_plotxy(i, j+5);
cursor2_Y += 10;

free_ivector(a);
}


//*************************************************************************
//****    The find peak and valley routines
//////////////////////////////////////////////////////////////////////////
// Subroutine to find next peak to the right in a projection
// -1 = none found
int find_peak_to_right(
  int proj[],           // The projection (an array)
  int proj_size,        // The max size of the projection (not inclusive)
  int pos,              // Where to start looking
  int min_peak_height)  // Minimum value for the peak
{
  int peak;             // Where the peak is

  peak = -1;            // Set to not found
  while ((++pos < proj_size) && (peak == -1))
    if ((proj[pos-1] >= proj[pos]) && (proj[pos-1] >= min_peak_height))
      peak = pos - 1;
  return peak;
}


//////////////////////////////////////////////////////////////////////////
// Subroutine to find next peak to the left in a projection
// -1 = none found
int find_peak_to_left(
  int proj[],           // The projection
  int pos,              // Where to start looking
  int min_peak_height)  // Minimum value for the peak
{
  int peak;             // Where the peak is
```

【図１７１】

```
  peak = -1;            // Set to not found
  while ((--pos >= 0) && (peak == -1))
    if ((proj[pos+1] >= proj[pos]) && (proj[pos-1] >= min_peak_height))
      peak = pos + 1;
  return peak;
}


////////////////////////////////////////////////////////////////////////
// Subroutine to find next valley to the right in a projection
// -1 = none found
int find_valley_to_right(
  int proj[],           // The projection
  int proj_size,        // The max size of the projection (not inclusive)
  int pos,              // Where to start looking
  int max_valley_depth) // Highest point a valley can be
{
  int valley;           // Where the valley is

  valley = -1;          // Set to not found
  while ((++pos < proj_size) && (valley == -1))
    if ((proj[pos-1] <= proj[pos]) && (proj[pos-1] <= max_valley_depth))
      valley = pos - 1;
  return valley;
}


////////////////////////////////////////////////////////////////////////
// Subroutine to find next valley to the left in a projection
// -1 = none found
int find_valley_to_left(
  int proj[],           // The projection
  int pos,              // Where to start looking
  int max_valley_depth) // Maximim value for the valley
{
  int valley;           // Where the peak is

  valley = -1;          // Set to not found
  while ((--pos >= 0) && (valley == -1))
    if ((proj[pos+1] <= proj[pos]) && (proj[pos+1] <= max_valley_depth))
      valley = pos + 1;
  return valley;
}


////////////////////////////////////////////////////////////////////////
//////    Compute the Mehrzad's angles (given a valley position)    //////
////////////////////////////////////////////////////////////////////////
void find_valley_angles(
  int* proj,            // The projection vector
  int proj_size,        // The size of the projection vector
  int valley_pos,       // The position to the right of the 1st peak
  double& theta1,       // Angle of left wall
  double& theta2,       // Angle of right wall
  int min_peak_height,  //
  int max_valley_depth)
{
  int p1x, p1y;         // The coord of left peak
  int p2x, p2y;         // The coord of left valley bottom
  int p3x, p3y;         // The coord of right valley
  int p4x, p4y;         // The coord if right peak

  // Find p1: the peak to the left of the valley
  p1x = find_peak_to_left(proj, valley_pos, min_peak_height);
  if (p1x == -1)        // If none found, make it the left side
    p1x = 0;
  p1y = proj[p1x];      // Get its height

  // Find p2: the valley to the right of the given peak
  p2x = find_valley_to_right(proj, proj_size, p1x, max_valley_depth);
  if (p1x == -1)        // If not found, make it the right side
    p2x = proj_size - 1;
  p2y = proj[p2x];

  // Find p4: the next peak to the right of valley p2
  p4x = find_peak_to_right(proj, proj_size, p2x, min_peak_height);
  if (p4x == -1)        // If not found, make it the right side
    p4x = proj_size - 1;
  p4y = proj[p4x];

  // Find p3: the nearest valley to the left of peak at p4
  p3x = find_valley_to_left(proj, p4x, max_valley_depth);
  if (p3x == -1)        // If not found, make it the left side
    p3x = 0;
  p3y = proj[p3x];

  theta1 = atan((double) (p2x-p1x)/ (double) (p1y-p2y)); // This is positive
  theta2 = atan((double) (p4x-p3x)/ (double) (p3y-p4y)); // This is negative
}


////////////////////////////////////////////////////////////////////////
//////          Mehrzad's optimal threshold finding routine       //////
////////////////////////////////////////////////////////////////////////
void compute_optimal_threshold(
    int* proj,          // The VPP (an array)
    int proj_size,      // The size of proj[]
    int level,          // The threshold level
    int& max_valley_depth,  // The outputs
    int&  min_peak_height)
{
  int max_value_proj;   // The maximum value of the projection
  int i;
  double a, b;          // Values
  FILE *fp;

  if ((fp = fopen("constants.dat", "r")) == NULL)
  {
    cerr << "\nError in opening constants.dat\n";
    exit(1);
  }

  fscanf(fp, "%lf", &a);            // Skip sin char width
  for (i=0; i<=level; i++)
  {
    fscanf(fp, "%lf%lf", &a, &b);   // Read a line
    if ((a <= -1) || (b <= -1))     // Exit
    {
      max_valley_depth = -1;        // Return an error
      min_peak_height = -1;
      return;
    }
  }
  fclose(fp);                       // Close the file

//  printf("*** LEVEL %d = %f, %f\n", level, a, b);

  // Find maximum projection (for find peak and valley heights)
  if (proj_size <= 0)               // Check for size > 0
  {
    max_valley_depth = 0;
    min_peak_height = 0;
    return;
  }
```

【図173】

```
max_value_proj = proj[0];
for (i=1; i<proj_size; i++)
  if (proj[i] > max_value_proj)
    max_value_proj = proj[i];

// Compute min and max valley heights
min_valley_depth = (int) (a * (double) max_value_proj);
min_peak_height = (int) (b * (double) max_value_proj);
}


////////////////////////////////////////////////////////////////////////
//////
//////    Reads in the maximum character width from the data file    //////
//////
////////////////////////////////////////////////////////////////////////
int get_max_char_width()
{
  FILE *fp;
  double i;

  if ((fp = fopen("constants.dat", "r")) == NULL)
  {
    cerr << "\nError in opening constants.datn";
    exit(1);
  }

  fscanf(fp, "%lf", &i);          // Read min char width
  fclose(fp);

  return i;
}


////////////////////////////////////////////////////////////////////////
//////
//////    Insert an angular break into the breaklist    //////
//////
////////////////////////////////////////////////////////////////////////
void insert_angular_break(
  Blist *blist,                   // The image
  Breaklist *blist,               // The breaklist
  int bnum1,                      // The angular range will be forced to
  int bnum2,                      //   fall within these breaks
  double theta,                   // The angle
  int pos)                        // The break position (for line computation)
{
  int break_position;             // The location of the break in the breaklist
  int *brk;                       // The new break
  int *a1, *a2;                   // Arrays, breaks for bnum1, and bnum2
  int i;

  brk = ivector(image.size0);     // Allocate memory for a2
  a1 = ivector(image.size0);      // Allocate memory for a1
  a2 = ivector(image.size0);      // Allocate memory for a2
  Breaklist_return_curves(blist, bnum1, bnum2, a1, a2); // Compute break range

  for (i=0; i<image.size1; i++)
  {
    // Compute break position
    brk[i] = (int) ((double) pos / cos(theta) + (double) i * tan(theta));

    // Ensure that it is within range
    if (brk[i] < a1[i])
      brk[i] = a1[i];
    if (brk[i] > a2[i])
      brk[i] = a2[i];
  }

  break_position = (brk[0] + brk[image.size1-1]) / 2;

  // Insert the break
  Breaklist_add_break(blist, (break_position*blist->bvector_height, BREAK_ANGULAR, b
nk);

  free_ivector(brk);              // Free memory
  free_ivector(a2);
  free_ivector(a1);
}


////////////////////////////////////////////////////////////////////////
//////
//////    Simple routine used by insert_angular_breaks()    //////
//////    Checks a value against the current minimum value.  If less, it updates
//////
////////////////////////////////////////////////////////////////////////
void insert_angular_compare_min(
  int min,                        // The current minimum
  int min_at,                     // Where it's at
  double min_theta,               // It's angle
  int tmin,                       // The test minimum
  int tmin_at,                    //   etc...
  double tmin_theta)
{
  if (tmin_at < 0)                // Don't substitute if tmin_at == -1
    return;
  if (tmin < min)
  {
    min = tmin;
    min_at = tmin_at;
    min_theta = tmin_theta;
  }
}


////////////////////////////////////////////////////////////////////////
//////
//////    Manual's angular projection break routine V2?    //////
//////
////////////////////////////////////////////////////////////////////////
// Inserts angular breaks in a segment
int insert_angular_breaks(
  image& image,                   // Returns number of breaks inserted
  Breaklist* blist,               // The image
  int start_break,                // The breaklist
  int stop_break,                 // The starting break number
  int thresh_level);              // The stopping break number
                                  // Which set of thresholds to use
{
  int *proj, proj_alloc;          // Projection vector and its size
  int proj_size, proj_zero;       // Highest used value, and what value zero is
  int max_value_proj;             // The maximum value of the projection
  int valley_pos;                 // Position of valley
  int max_valley_depth;           // Maximum height for a valley
  int min_peak_height;            // Minimum height for a peak
  int i, j, i2;                   // Misc var
  double theta1, theta2;          // Angles of valley walls
  int break_count;
  int min, min_at;                // Used for finding angular projections
  double min_theta;               // A minimum and where it's located,
  int tmin, tmin_at;              // its angle
                                  // A temporary min and where it's located

  // Zero number of breaks
  break_count = 0;

  // Allocate the projection vector
  proj_alloc = image.size2();     // This should be more than enough
  proj = ivector(proj_alloc);
```

【図174】

```
// Find vertical projection
find_projection(image, 0, blist, start_break, stop_break, image.size(),
                proj_size, proj, proj_size, proj_zero); // Compute VPP

// Find maximum projection (for find peak and valley heights)
max_value_proj = proj[0];
for (i=1; i<proj_size; i++)
  if (proj[i] > max_value_proj)
    max_value_proj = proj[i];

// Compute max_valley_depth & min_peak_height
compute_optimal_threshold(proj, proj_size, thresh_level,
  max_valley_depth, min_peak_height);
if (max_valley_depth < 0)      // Exit if we're too deep in constants
  return break_count;

#ifdef VISIBLE
// ***** PLOT *****
cursor1_x = cursor1_y = 20*thresh_level;          // Initialize Cursor2 (going down)
cursor2_y = cursor1_y + 20*thresh_level;          // Compute L and R columns
struct Break bl, br;
Breaklist_return_break(blist, start_break, &bl);
Breaklist_return_break(blist, stop_break, &br);
cursor1_x = (br.loc - bl.loc)/image.size.x * 40; // Compute width of column
cursor_col_left = cursor_col_right;   //   (and move cursor 1 to right)
cursor_col_right = cursor1_x-10;
for (i=cursor_top_dline; i<WINDOW_SIZE_Y; i++) // Plot double vert. column
  {
  xs_plotxy(cursor_col_left-1, i);
  xs_plotxy(cursor_col_left+1, i);
  }
print_image_with_breaks(image, blist, start_break, stop_break,
    cursor1_x, cursor1_y);
cursor1_x -= image.size.x + 7;
xs_flush();
//***** ENDPLOT ****

// ***** PLOT *****, Plot the histogram
int ttt = height_plot(proj, proj_size, cursor2_x+2, cursor2_y);
int ccol1 = cursor1_x + ttt - min_peak_height;    // Plot thresholds
int ccol2 = cursor1_x + ttt - max_valley_depth;
for (i=0; i<proj_size-5; i++)
  if (i % 2)
    xs_plotxy(i+cursor1_x, ccol1);
  else xs_plotxy(i+cursor1_x, ccol2);
int ttemp_cursor1_y = cursor1_y + ttt + 4;
cursor1_y -= ttt + 10;
xs_flush();
//***** ENDPLOT ****
#endif

// Find first peak to right
i = find_peak_to_right(proj, proj_size, 0, min_peak_height);
if (i == -1)                          // If peak not found...
  return break_count;

// Find each valley, starting from left position
while(1)
{
  // Search for valley to right
  // Note: i should be the peak on the left!
  valley_pos = find_valley_to_right(proj, proj_size, i, max_valley_depth);
  if (valley_pos == -1)               // If not found
    return break_count;

  // Search for next peak to right of valley_pos
  i = find_peak_to_right(proj, proj_size, valley_pos, min_peak_height);
  if (i == -1)                        // If not found
    return break_count;

#ifdef VISIBLE
// ***** PLOT ***** ***** DELETE WE EVENTUALLY ****** (plot tick for break)
int ttt = find_peak_to_left(proj, valley_pos, min_peak_height);
for (int j=ttt; j<i; j++)
  xs_plotxy(j+cursor2_x-2, ttemp_cursor_y-2);
xs_plotxy(ttt+cursor2_x-2, ttemp_cursor_y-2);
xs_plotxy(i+cursor2_x-2, ttemp_cursor_y-3);
xs_plotxy(i+cursor2_x+2, ttemp_cursor_y-3);
xs_flush();
//***** ENDPLOT ****
#endif

  // Now we know a valley exists, so compute angles
  find_valley_angles(proj, proj_size, valley_pos, theta2,
                     min_peak_height, max_valley_depth);

//  cout << 'Position ' << valley_pos.proj_zero << ': Angles = ' << theta1*180/3.1
159 << ', ' << theta2*180/3.14159 << '\n';

  // Place C at the next peak (on the right)
  C = i-proj_zero;

  // Find minimum of the VPP (theta = 0)
  find_minimum_at_vector(proj, valley_pos, C-proj_zero, min, min_at);
  min_theta = 0;
  min_at += proj_zero;

//  cout << '  VPP minimum at ' << min_at << ' = ' << min << '\n';

  // Compute the angular vpp from the last break to C, and find minimums
  find_min_valley_at_theta(image, blist, start_break+break_count,
                           stop_break+break_count, C, theta1, tmin, tmin_at);

//  cout << '    At Theta1, min = ' << tmin << '\n';
  // If it has a smaller minimum, remember it
  insert_angular_compare_min(min, min_at, min_theta, tmin, tmin_at, theta1);

  find_min_valley_at_theta(image, blist, start_break+break_count,
                           stop_break+break_count, C, theta2, tmin, tmin_at);

//  cout << '    At Theta2, min = ' << tmin << '\n';
  insert_angular_compare_min(min, min_at, min_theta, tmin, tmin_at, theta2);

  ////////////////// Check other angles;
  double three_degrees = 3.14159/180.0 * 3.0;
  double six_degrees = 3.14159/180.0 * 6.0;
  double nine_degrees = 3.14159/180.0 * 9.0;

  find_min_valley_at_theta(image, blist, start_break+break_count,
                           stop_break+break_count, C, theta1+three_degrees, tmin, tmin_at);
//  cout << '    At Theta1+3, min = ' << tmin << '\n';
  insert_angular_compare_min(min, min_at, min_theta, tmin, tmin_at,
                             theta1+three_degrees);

  // Check theta1 - 3
  find_min_valley_at_theta(image, blist, start_break+break_count,
                           stop_break+break_count, C, theta1-three_degrees, tmin, tmin_at);
//  cout << '    At Theta1-3, min = ' << tmin << '\n';
  insert_angular_compare_min(min, min_at, min_theta, tmin, tmin_at,
                             theta1-three_degrees);

  // Check theta1 + 6
  find_min_valley_at_theta(image, blist, start_break+break_count,
                           stop_break+break_count, C, theta1+six_degrees, tmin, tmin_at);
```

【図175】

```
//    cout << "  At Theta1-6, min = " << min << '\n';
//    insert_angular_compare_min(min, min_at, min_theta, tmin, tmin_at,
                                 theta1+six_degrees);

      // Check theta1 - 6
      find_min_valley_at_theta(image, blist, start_break+break_count,
          stop_break+break_count, C, theta1-six_degrees, tmin, tmin_at);
        cout << "  At Theta1-6, min = " << tmin << '\n';
//      insert_angular_compare_min(min, min_at, min_theta, tmin, tmin_at,
                                   theta1-six_degrees);

      // Check theta1 + 6
      find_min_valley_at_theta(image, blist, start_break+break_count,
          stop_break+break_count, C, theta1+nine_degrees, tmin, tmin_at);
        cout << "  At Theta1+6, min = " << tmin << '\n';
//      insert_angular_compare_min(min, min_at, min_theta, tmin, tmin_at,
                                   theta1+nine_degrees);

      // Check theta1 - 6
      find_min_valley_at_theta(image, blist, start_break+break_count,
          stop_break+break_count, C, theta1-nine_degrees, tmin, tmin_at);
        cout << "  At Theta1-6, min = " << tmin << '\n';
//      insert_angular_compare_min(min, min_at, min_theta, tmin, tmin_at,
                                   theta1+three_degrees);

      // Check theta2 - 3
      find_min_valley_at_theta(image, blist, start_break+break_count,
          stop_break+break_count, C, theta2+three_degrees, tmin, tmin_at);
        cout << "  At Theta2-3, min = " << tmin << '\n';
//      insert_angular_compare_min(min, min_at, min_theta, tmin, tmin_at,
                                   theta2-three_degrees);

      // Check theta2 - 6
      find_min_valley_at_theta(image, blist, start_break+break_count,
          stop_break+break_count, C, theta2+six_degrees, tmin, tmin_at);
        cout << "  At Theta2-6, min = " << tmin << '\n';
//      insert_angular_compare_min(min, min_at, min_theta, tmin, tmin_at,
                                   theta2-six_degrees);

      // Check theta2 + 6
      find_min_valley_at_theta(image, blist, start_break+break_count,
          stop_break+break_count, C, theta2-six_degrees, tmin, tmin_at);
        cout << "  At Theta2+6, min = " << tmin << '\n';
//      insert_angular_compare_min(min, min_at, min_theta, tmin,
                                   theta2+six_degrees);

      // Check theta2 - 6
      find_min_valley_at_theta(image, blist, start_break+break_count,
          stop_break+break_count, C, theta2-nine_degrees, tmin, tmin_at);
        cout << "  At Theta2-6, min = " << tmin << '\n';
//      insert_angular_compare_min(min, min_at, min_theta, tmin,
                                   theta2-nine_degrees);

//    // Insert the best break
//      cout << "  Final at " << min_at << ", theta = " << min_theta*180/3.14159 << ',
//    value = " << min << '\n';

      tmin_at = min_at;
#ifdef VISIBLE
      find_min_valley_at_theta_print(image, blist, start_break+break_count,
          stop_break+break_count, C, min_theta, tmin, tmin_at);
#endif
}


insert_angular_break(image, blist, start_break+break_count,
    stop_break+break_count, min_theta, min_at);

  break_count++;          // Move 1 to the next peak do another peak
}


////////////////////////////////////////////////////////////////////////
// This routine is used as a link to pass a character to recognition
// routine if need to go to deeper level. It returns a
// '0' to go deeper, '1' otherwise (if know, it's backward)
//
int pass_to_deeper_level(
    gimage& image,          // The image
    Breaklist* blist,       // The breaklist
    int b1,
    int b2)
{
  int **a;                  // The matrix to send to the recognition routine
  int size_r, size_c;       // The size of matrix a
  int i,j;
  int return_value;         // Used to calculate size_c
  int i,j;
  int *array1, *array2;

  array1 = ivector(image.sizex);
  array2 = ivector(image.sizex);

  Breaklist_return_curves(blist, b1, b2, array1, array2);

  r = -1; l = 31000;
  for (i=0; i<image.sizex; i++)
  {
    if (array1[i] < l)  // Get range
      l = array1[i];
    if (array2[i] < l)
      l = array2[i];
    if (array1[i] > r)
      r = array1[i];
    if (array2[i] > r)
      r = array2[i];
  }
  size_c = r - l;
  size_r = image.sizex;

  if (size_r > MAX_CHAR_WIDTH)  // If too wide, automatic failure
    return 0;

  a = imatrix(size_r, size_c);

  for (i=0; i<size_r; i++)
    for (j=0; j<size_c; j++)
      a[i][j] = 0;

  for (i=0; i<image.sizex; i++)
    for (j=array1[i]; j<array2[i]; j++)
      if (image.pixel[i][j])
        a[i][j-l] = 255;

  return_value = recognised(a, size_r, size_c);

  free_imatrix(a, size_r);
  return return_value;
}
```

```
////////////////////////////////////////////////////////////////////////////
////    Recursivly insert breaks with higher and higher break levels    ////
////          (Mehrzad's "goodcut" routine (I didn't name it))          ////
////////////////////////////////////////////////////////////////////////////
int layer4_goodcut(                 // Returns the number of breaks added
  gimage& image,                    // The image
  Breaklist* blist,                 // The breaklist
  int bnum1,                        // The starting break number
  int bnum2,                        // The stopping break number
  int thresh_level,                 // Which set of thresholds to use
  int perform_recombining)          // == 1 to do recombining, 0 otherwise
{
  int b;                            // Number of breaks added in this routine
  struct Break *brk, *brk2;         // A pointer to the break
  int i,j;
  int R[MAX_CHARS_PER_CHUNK];       // Which piece is recognized, and it's
                                    // corresponding break
  int bn, br;                       // Used to represent start of break in goodcut

  printf("Goodcut Level %d\n", thresh_level);

  // Check to see if width is too long
  brk = Breaklist_return_break_ptr(blist, bnum1);
  brk2 = Breaklist_return_break_ptr(blist, bnum2);
  if ((brk2->loc - brk->loc) < MIN_CHAR_WIDTH * image.sizeR)
  {
    printf("    Goodcut stoped going up layers because width too small");
    return 0;
  }

  // Insert the breaks
  b = 0;                            // Initialize
  while (b == 0)                    // Loop until break is added
  {
    compute_optimal_threshold(0, 0, thresh_level, i, i);
    if (i <= -1)                    // If there are no more threshold values to try,
    {
      printf("    Max level reached.  Returning down.\n");
      return 0;                     // Exit and don't go deeper
    }

    // Insert breaks in the image
    b = insert_angular_breaks(image, blist, bnum1, bnum2, thresh_level);
    printf("    %d breaks inserted at level %d.\n", b, thresh_level);
    thresh_level++;
  }

  // Pass each piece which is > MAX_LENGTH to deeper levels
  printf("Passing each piece that is too long to level.\n");
  for (i=bnum1; i<bnum2+b; i++)     // Warning: Deceptive for loop
  {
    brk = Breaklist_return_break_ptr(blist, i);
    brk2 = Breaklist_return_break_ptr(blist, i+1);
    if ((brk2->loc - brk->loc) > MAX_CHAR_WIDTH * image.sizeR)
    {
      printf("   A break was passed to a higher level because too long\n");
      j = layer4_goodcut(image, blist, i, i+1, thresh_level, 0);
      if (j != 0)                   // If didn't bottom out,
        i += 1;                     // Check each of the new pieces
      b += j;
    }
  }

  if (b == 0)                       // If no breaks inserted
  {
    printf("Couldn't find a place to insert a break.  Exiting goodcut.\n");
    return b;                       // Exit (gone to highest level and can't cut)
  }

  ////////////////////////////
  // GOODCUT METHOD 2 FOLLOWS:  (The recombining)

  if (perform_recombining == 0)
    return b;

  // Pass each piece to recognition  (breaklist range = bnum1 to bnum2+b)

  // Test to see if MAX_CHARS_PER_CHUNK is high enough
  if (bnum2-bnum1+b+1 >= MAX_CHARS_PER_CHUNK)   //
  {
    printf("MAX_CHARS_PER_CHUNK in multilevel.C is too small!\n");
    printf("   In this case it needs to be %d\n", bnum2-bnum1+b+1);
    exit(1);
  }

  // Load in R[], the recognition array: 0 = Not Recognized, 1 = Recognized,
  // and -1 = end of list
  for (i=bnum1; i<bnum2+b; i++)  // Loop through each section
  {
    brk = Breaklist_return_break_ptr(blist, i);
    if (brk->next_recog == -1)  // If not previously recognized
      R[i-bnum1] = brk->next_recog = pass_to_deeper_level(image, blist, i, i+1);
  }
  R[bnum2+bnum1+b] = -1;        // End of the Breaklist

  printf("Recognize List:\n  E ");
  i = 0;
  while(R[i] != -1)
  {
    if (R[i] == 0)
      printf("N ");
    else
      printf("Y ");
    i++;
  }
  printf("E\n");

  printf("Goodcut Logic: \n");
  if (R[0] == 0)                // If first not recognized
  {                             ////////////////////// ON LEFT EDGE ////////////////
    printf("  E N\n");
    if (R[1] == 0)                                          // If NN
    {
      printf("  E N N\n");
      printf("  Trying E [N N] = ");
      if (pass_to_deeper_level(image, blist, bnum1, bnum1+2))   // Try 0-1
      {
        printf("Good\n");
        Breaklist_remove_break(blist, bnum1+1);             // 0-1 Good
        b -= 1;
        brk = Breaklist_return_break_ptr(blist, bnum1);
        brk->next_recog = 1;
        return b;
      }
      printf("Bad\n");
      if (R[2] == -1)                                       // If NNE
      {
        printf("  E N N E - Combining E [N N] E and passing lower\n");
        Breaklist_remove_break(blist, bnum1+1);             // Pass 0-1 Deeper
        brk = Breaklist_return_break_ptr(blist, bnum1);
        brk->next_recog = -1;
        b -= 1;
        b += layer4_goodcut(image, blist, bnum1, bnum1+1, thresh_level, 1);
```

【図177】

```
    return b;
  }
  if (R[3] == 0)                        // If NNN
  {
    printf("  E N N Nin");
    printf("  Trying E N N [N N] - \n");
    if (pass_to_deeper_level(image, blist, bnum1+1, bnum1+2)) // Try 1-2
    {
      printf("Good\n");
      Breaklist_remove_break(blist, bnum1+2);       // Combine 1-2
      b -= 1;
      brk = Breaklist_return_break_ptr(blist, bnum1+1);
      brk->next_recog = 1;
      return b;
    }
    printf("Bad\n");
    printf("  Trying E [N N N] - ");
    if (pass_to_deeper_level(image, blist, bnum1, bnum1+2)) // Try 0-2
    {
      printf("Good\n");
      Breaklist_remove_break(blist, bnum1+1);
      Breaklist_remove_break(blist, bnum1+1);
      b -= 2;
      brk = Breaklist_return_break_ptr(blist, bnum1);
      brk->next_recog = 1;
      return b;
    }
    printf("Bad.\n  Combining E [N N N] E  and passing deeper\n");
    Breaklist_remove_break(blist, bnum1+1);     // Pass 0-2 deeper
    Breaklist_remove_break(blist, bnum1+1)
    b -= 2;
    b += layer4_goodcut(image, blist, bnum1, bnum1+1, thresh_level, 1);
    return b;
  }
  if (R[1] == 1)                        // If NY
  {
    printf("  E N Yin");
    printf("  Trying E [N Y] - ");
    if (pass_to_deeper_level(image, blist, bnum1+1))    // Try 0-1
    {
      printf("Good\n");
      Breaklist_remove_break(blist, bnum1+1);       // 0-1 Good
      b -= 1;
      brk = Breaklist_return_break_ptr(blist, bnum1);
      brk->next_recog = 1;
      return b;
    }
    printf("Bad\n");
    printf(" Combing E [N Y] and passing deeper\n");
    Breaklist_remove_break(blist, bnum1+1);     // Pass 0-1 Deeper
    b -= 1;
    brk = Breaklist_return_break_ptr(blist, bnum1);
    brk->next_recog = -1;
    b += layer4_goodcut(image, blist, bnum1, bnum1+1, thresh_level, 1);
    return b;
  }
  if ((R[1] == 0) && (R[2] == 1))       // If NNY
  {
    printf("  E N E Yin  Trying E [N N Y] - ");
    if (pass_to_deeper_level(image, blist, bnum1, bnum1+2)) // Try 0-1
    {
      printf(" Good\n");
      Breaklist_remove_break(blist, bnum1+1);
      Breaklist_remove_break(blist, bnum1+1);
      b -= 2;
      brk = Breaklist_return_break_ptr(blist, bnum1);
      brk->next_recog = 1;
      return b;
    }
    printf("Bad.\n");
    printf("  Combining E [N N Y] and passing deeper\n");
    Breaklist_remove_break(blist, bnum1+1);     // Pass 0-2 deeper
    b -= 2;
    brk = Breaklist_return_break_ptr(blist, bnum1);
    brk->next_recog = -1;
    b += layer4_goodcut(image, blist, bnum1+1, thresh_level, 1);
    return b;
  }
  if ((R[1] == 0) && (R[2] == 0))       // If NNN
  {
    printf("  E N E Nin  Trying E [N N N] - ");
    if (pass_to_deeper_level(image, blist, bnum1+1, bnum1+2))   // Try 0-1
    {
      printf(" Good\n");
      Breaklist_remove_break(blist, bnum1+1);
      Breaklist_remove_break(blist, bnum1+1);
      b -= 2;
      brk = Breaklist_return_break_ptr(blist, bnum1);
      brk->next_recog = 1;
      return b;
    }
    printf("Bad\n");
    printf("  Combining E [N N N] and passing deeper\n");
    Breaklist_remove_break(blist, bnum1+1);     // Pass 0-2 deeper
    b -= 2;
    brk = Breaklist_return_break_ptr(blist, bnum1);
    brk->next_recog = -1;
    b += layer4_goodcut(image, blist, bnum1, bnum1+2, thresh_level, 1);
    return b;
  }
  if ((R[1] == 1) && (R[2] == 1))       // If YYY
  {
    printf("  E Y E Yin  Trying E [Y Y Y] - ");
    if (pass_to_deeper_level(image, blist, bnum1+1, bnum1+2))   // Try 0-2
    {
      printf(" Good\n");
      Breaklist_remove_break(blist, bnum1+1);
      Breaklist_remove_break(blist, bnum1+1);
      b -= 2;
      brk = Breaklist_return_break_ptr(blist, bnum1);
      brk->next_recog = 1;
      return b;
    }
    printf("Bad\n");
    printf("  Combining E [Y Y Y] and passing deeper\n");
    Breaklist_remove_break(blist, bnum1+1);     // Pass 0-2 deeper
    b -= 2;
    brk = Breaklist_return_break_ptr(blist, bnum1);
    brk->next_recog = -1;
    b += layer4_goodcut(image, blist, bnum1+1, thresh_level, 1);
    return b;
  }
  if ((R[1] == 1) && (R[2] == 0))       // If YYN
  {
    printf("  E Y E Nin  Trying E [Y Y N] - ");
    if (pass_to_deeper_level(image, blist, bnum1+1, bnum1+2))   // Try 0-1
    {
      printf(" Good\n");
      Breaklist_remove_break(blist, bnum1+1);
      Breaklist_remove_break(blist, bnum1+1);
      b -= 2;
      brk = Breaklist_return_break_ptr(blist, bnum1);
      brk->next_recog = 1;
      return b;
    }
    printf("Bad\n");
    printf("  Combining E [Y Y N] and passing deeper\n");
    Breaklist_remove_break(blist, bnum1+1);     // Pass 0-2 deeper
    b -= 2;
    brk = Breaklist_return_break_ptr(blist, bnum1);
    brk->next_recog = -1;
    b += layer4_goodcut(image, blist, bnum1+1, thresh_level, 1);
    return b;
  }
  if ((R[1] == 0) && (R[2] == 1))       // If NNY
  {
    printf("  E N E Yin  Trying E [N N Y] - ");
    if (pass_to_deeper_level(image, blist, bnum1, bnum1+2)) // Try 0-1
    {
      printf(" Good\n");
      Breaklist_remove_break(blist, bnum1+1);
      Breaklist_remove_break(blist, bnum1+1);
      b -= 2;
      brk = Breaklist_return_break_ptr(blist, bnum1);
      brk->next_recog = 1;
      return b;
    }
    printf("Bad.\n");
    printf("  Combining E [N N Y] and passing deeper\n");
    Breaklist_remove_break(blist, bnum1+1);     // Pass 0-2 deeper
    b -= 2;
    brk = Breaklist_return_break_ptr(blist, bnum1);
    brk->next_recog = -1;
    b += layer4_goodcut(image, blist, bnum1+1, thresh_level, 1);
    return b;
  }
  if ((R[1] == 0) && (R[2] == 0))       // If NNN
  {
    printf("  E N E Nin  Trying E [N N N] - ");
    if (pass_to_deeper_level(image, blist, bnum1+1, bnum1+2))   // Try 0-1
    {
      printf(" Good\n");
      Breaklist_remove_break(blist, bnum1+1);
      Breaklist_remove_break(blist, bnum1+1);
      b -= 2;
      brk = Breaklist_return_break_ptr(blist, bnum1);
      brk->next_recog = 1;
      return b;
    }
    printf("Bad\n");
    printf("  Combining E [N N N] and passing deeper\n");
    Breaklist_remove_break(blist, bnum1+1);     // Pass 0-2 deeper
    b -= 2;
    brk = Breaklist_return_break_ptr(blist, bnum1);
    brk->next_recog = -1;
    b += layer4_goodcut(image, blist, bnum1, bnum1+2, thresh_level, 1);
    return b;
  }
  printf("  All breaks recognized.  Returning.\n");
  return b;
}
bn--;
bn = br+bnum1;                          // (Start with the Y before the N
                                        // b = 0 corresponds to brake bn
printf("Found ... Y N:\n");
printf("  Trying [Y N] - ");
if (pass_to_deeper_level(image, blist, bn, bn+2))   // Try 0-1
{
  printf("Good\n");
  Breaklist_remove_break(blist, bn+1);              // 0-1 Good
  b -= 1;
  return b;
}
if (R[br+2] == -2)                      // We have YN
{
  printf("  Y N Bin");
  printf("  Combining [Y N] E and passing lower\n");   // Pass 0-1 Deeper
  b--;
  brk = Breaklist_return_break_ptr(blist, bn);
  brk->next_recog = -2;
  b += layer_goodcut(image, blist, bn, bn+2, thresh_level, 1);
  return b;
}
if (R[br+2] == 0)                       // We have YNN
{
  printf("  Y N Nin");
  printf("  Trying Y [N N] - ");
  if (pass_to_deeper_level(image, blist, bn+1, bn+3))
  {
    printf("Good\n");
    Breaklist_remove_break(blist, bn+2);            // 1-2 Good
    b--;
    brk = Breaklist_return_break_ptr(blist, bn+1);
    brk->next_recog = 1;
    return b;
  }
  printf("Bad\n Trying [Y N N] - ");
  if (pass_to_deeper_level(image, blist, bn+1, bn+3))
  {
    printf("Good\n");                               // 0-2 Good
    Breaklist_remove_break(blist, bn+1);
    Breaklist_remove_break(blist, bn+1);
    b-=2;
    return b;
  }
  printf("Bad\n");
  if (R[br+3] == -2)
```

【図178】

```
{
  printf(" Y N N Y\n");                                  // YNNE
  printf(" Combining [Y N N] E and passing deeper\n");
  BreakList_remove_break(blist, bn+1);                   // Pass 0-2 Deeper
  BreakList_remove_break(blist, bn+1);
  b-=2;
  brk = BreakList_return_break_ptr(blist, bn);
  brk->next_recog = -1;
  b += layer4_goodcut(image, blist, m, bn+1, thresh_level, 1);
  return b;
}
if (R[br+3] == 0)                                        // YNNN
{
  printf(" Y N N N\n");
  printf(" Trying Y [N N] - ");
  if (pass_to_deeper_level(image, blist, bn+2, bn+4))
  {
    printf("Good\n");
    BreakList_remove_break(blist, bn+3);                 // 2-3 Good
    b--;
    brk = BreakList_return_break_ptr(blist, bn+2);
    brk->next_recog = 1;
    return b;
  }
  printf("Bad\n  Trying Y [N N N] - ");
  if (pass_to_deeper_level(image, blist, bn+1, bn+4))
  {
    printf("Good\n");
    BreakList_remove_break(blist, bn+2);                 // 1-3 Good
    BreakList_remove_break(blist, bn+2);
    b-=2;
    brk = BreakList_return_break_ptr(blist, bn+1);
    brk->next_recog = 1;
    return b;
  }
  printf("Bad\n  Trying [Y N N N] - ");
  if (pass_to_deeper_level(image, blist, bn, bn+4))
  {
    printf("Good\n");
    BreakList_remove_break(blist, bn+1);                 // 0-3 Good
    BreakList_remove_break(blist, bn+1);
    BreakList_remove_break(blist, bn+1);
    b-=3;
    return b;
  }
  printf("Bad\n Combining Y [N N N] and passing to deeper level\n");
  BreakList_remove_break(blist, bn+2);                   // Pass 1-3 Deeper
  BreakList_remove_break(blist, bn+2);
  b-=2;
  brk = BreakList_return_break_ptr(blist, bn+1);
  brk->next_recog = -1;
  b += layer4_goodcut(image, blist, bn+1, thresh_level, 1);
  return b;
}
if (R[br+2] == 1)                                        // We have YNY
{
  printf(" Y N Y\n");
  printf(" Trying Y [N Y] - ");
  if (pass_to_deeper_level(image, blist, bn+1, bn+3))
  {
    printf("Good\n");
    BreakList_remove_break(blist, bn+2);                 // 1-2 Good
    b--;
    brk = BreakList_return_break_ptr(blist, bn+1);
    brk->next_recog = 1;
    return b;
  }
  printf(" Combining [Y N N Y] and passing deeper\n");   // Pass 0-2 Deeper
  BreakList_remove_break(blist, bn+1);
  BreakList_remove_break(blist, bn+1);
  b-=2;
  brk = BreakList_return_break_ptr(blist, bn);
  brk->next_recog = -1;
  b += layer4_goodcut(image, blist, bn, bn+1, thresh_level, 1);
  return b;
}
if ((R[br+2] == 0) && (R[br+3] == 1))                    // YNNY
{
  printf(" Y N N Y\n");
  printf(" Trying Y [N N Y] - ");
  if (pass_to_deeper_level(image, blist, bn+2, bn+4))
  {
    printf("Good\n");
    BreakList_remove_break(blist, bn+2);                 // 1-3 Good
    BreakList_remove_break(blist, bn+2);
    b-=2;
    brk = BreakList_return_break_ptr(blist, bn+1);
    brk->next_recog = 1;
    return b;
  }
  printf(" Combining [Y N N Y] and passing deeper\n");   // Pass 0-3 Deeper
  BreakList_remove_break(blist, bn+1);
  BreakList_remove_break(blist, bn+1);
  BreakList_remove_break(blist, bn+1);
  b-=3;
  brk = BreakList_return_break_ptr(blist, bn);
  brk->next_recog = -1;
  b += layer4_goodcut(image, blist, bn, bn+1, thresh_level, 1);
  return b;
}
printf("WARNING: No pattern matched the goodcut routines.\n");
return b;
}

////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////
void BreakList_segmentation(
       CImage& image,          // The input image, a gray image
       BreakList* blist)       // Space allocated for a break list
                               //   (this will be initialized)
{
  int first_break, last_break;  // Misc. Used for break numbers
  int *proj, *proj_alloc;       // Projection vector and its size
  int b;
  struct Break *brk;

  // Allocate the projection vector
  proj_alloc = image.dimx()+1;
  proj = ivector(proj_alloc);

  // Find first and last breaks
  first_break = 0;
  last_break = blist->blist_count-1;

  if (last_break != 1)
  {
```

```
    printf("Warning: master_segmentation() (level 4) has somehow been\n");
    printf("         previously segmented!\n");
  }

  // Recognize this piece
  if (pass_to_deeper_level(image, blist, first_break, last_break) != 0)
                        // If recognized
  {
    brk = Breaklist_return_break_ptr(blist, 0);
    brk->next_recog = 1;
    free_ivector(proj); // Deallocate memory before exiting
    return;             //  then exit
  }

  // Call the recursive routine, starting at level 0, with recombining
  b = layer4_goodcut(image, blist, first_break, last_break, 0, 1);

  free_ivector(proj);

}


//////////////////////////////////////////////////////////////////
void print_image_with_breaks(
  ginagaa image,                // The image
  Breaklist* blist,             // The breaklist
  int break1,                   // The range of breaks to print
  int break2,
  int x,                        // The position
  int y)
{
  int r, c, b;
  int start_x = x;              // Starting value of x
  int *start, *stop;            // Start & Stop arrays
  struct Break b1;              // The info on a break
  int last_break;
  int wrap_x;

  wrap_x = WINDOW_SIZE_X - 50;

  if ((break1 < 0) || (break1 >= blist->blist_count) ||
      (break2 < 0) || (break2 >= blist->blist_count))
  {
    cerr << "Error in print_image_with_breaks: Illegal break range:\n";
    cerr << "Selected range is from " << break1 << " to " << break2 << "\n";
    cerr << "Break numbers range from 0 to " << blist->blist_count << "\n";
    int *null = 0;
    (*null) = 1;
    exit(1);
  }

  start = ivector(image.sizeR);
  stop = ivector(image.sizeR);

  // Find initial offset
  Breaklist_return_break(blist, break1, &b1);   // Get first break
  x -= b1.loc/image.sizeR;      // loc = the left side of the printing

  x-=2; y+=2;   // Move over a little for box boundry
  last_break = break1;
  for (b=break1+1; b <= break2; b++) // For each segment
  {
    Breaklist_return_break(blist, b, &b1);      // Get the break
    if (b1.type == BREAK_ERASED)        // Skip erased breaks!
      continue;
    Breaklist_return_curves(blist, last_break, b, start, stop);
    last_break = b;                     // Remember for next time

  for (r=0; r<image.sizeR; r++)
    for (c=start[r]; c<stop[r]; c++)
    {
      if (image.pixel[r][c] != 0)
        xs_plotxy(x+c,y+r);
      // Draw boarder
      if (r == 0)               // Draw top edge
        xs_plotxy(x+c, y+r-2);
      if (r == image.sizeR-1) // Draw bottom edge
        xs_plotxy(x+c, y+r+2);
      if (c == start[r])        // Draw left edge
        xs_plotxy(x+c-2, y+r);
      if (c == stop[r]-1)       // Draw right edge
        xs_plotxy(x+c+2, y+r);
      if ((r == 0) && (c == start[r]))          // Top left corner
      {
        xs_plotxy(x+c-1, y+r-1);
        xs_plotxy(x+c-2, y+r-2);
        xs_plotxy(x+c-1, y+r-1);
      }
      if ((r == 0) && (c == stop[r]-1))         // Top right corner
      {
        xs_plotxy(x+c+1, y+r-2);
        xs_plotxy(x+c+2, y+r-2);
        xs_plotxy(x+c+1, y+r-1);
      }
      if ((r == image.sizeR-1) && (c == stop[r]-1))  // Bottom right corner
      {
        xs_plotxy(x+c+1, y+r+2);
        xs_plotxy(x+c+2, y+r+2);
        xs_plotxy(x+c+2, y+r+1);
      }
      if ((r == image.sizeR-1) && (c == start[r]))    // Bottom left corner
      {
        xs_plotxy(x+c-1, y+r+2);
        xs_plotxy(x+c-2, y+r+2);
        xs_plotxy(x+c-2, y+r+1);
      }
    }
    if (x+c > wrap_x)
    {
      x -= wrap_x - start_x - 2;
      y += image.sizeX + 10;
    }
    x += 35;    // Slide over to next break is a little to right
  }

  free_ivector(stop);
  free_ivector(start);
}


//////////////////////////////////////////////////////////////////
void breaklist_to_charlist(
  ginage& line,                 // The picture of the entire line
  struct Breaklist *blist,              // The breaklist (for this chr)
  struct Charlist *clist,       // The characterlist we're processing
  struct Character **chr,       // The current character we're processing
  int left_side)                // The horiz distance between line and image
{
  struct Character *newchar;    // The new character.
  int i,b;

  if (blist->blist_count < 3)   // If no breaks
    return;
```

【図180】

```
for (b=1; b < blist->blis:_count-1; b++)
  {
    newchar = &(clist->chr[clist->size]);  // Get newchar to point to the end
                                           // of the charlist (the next blank character)

    // Copy curve2 of old char to curve2 of new char
    for (i=(*chr)->top; i<(*chr)->bot; i++)
      newchar->curve2[i] = (*chr)->curve2[i];
    // Copy break into curve2 of old char, and curve1 of new char
    Breaklist_return_curves(blist, b, b, &((*chr)->curve2[(*chr)->top]),
                            &(newchar->curve1[(*chr)->top]));

    // Slide over (due to image it's a sub-set of line)
    for (i=(*chr)->top; i<(*chr)->bot; i++)
      {
        newchar->curve1[i] += left_side;
        (*chr)->curve1[i] -= left_side;
        if (newchar->curve1[i] > newchar->curve1[i])
          newchar->curve1[i] = newchar->curve1[i];
        if ((*chr)->curve1[i] > (*chr)->curve1[i])
          (*chr)->curve1[i] = (*chr)->curve2[i];
      }

    newchar->top = (*chr)->top;
    newchar->bot = (*chr)->bot;

    Charlist_place_new_char(clist);     // Place newchar into the charlist
    ll2_shrinkwrap_char(line, chr);
    ll2_shrinkwrap_char(line, &newchar);
    *chr = newchar;
  }
}

//.....................................................................
//.....................................................................
//.....                      MAIN                               .....
//.....                                                         .....
//.....................................................................
//.....................................................................
void multi_level(
  cimage& line,
  struct Charlist *clist)       // The entire line
                                // The charlist
{
  gimage image;                 // A gimage of the character we're segmenting
  struct Breaklist blist;
  struct Break *brk;            // A pointer to the break
  cimage &l, &i;
  int dummy;
  int r,c,i,j;
  int size_x, size_y;           // The size of the character array
  struct Character *chr;        // the character we're processing

  chr = clist->first;           // Point to first character
  while(chr != 0)               // Loop through entire list
    {
#ifdef VISIBLE
    xs_clr_win();
    print_last_charlist(line, chr, 10, 10);
    xs_flush();
#endif
  printf("Enter 0 to pass char to level 4, or number of chars to skip? ");
  i = auto_input_int();
  if (i != 0)
    {
      for (j=0; j<i; j++)
        if (chr != 0)
          chr=chr->next;
      } else {

      printf("Doing level 4 processing\n");
      size_r = chr->bot - chr->top;
      size_c = chr->right - chr->left;  // Allocate size of matrix
      image.fill(0, size_r, size_c);    // Allocate and zero image

      for (r=chr->top; r < chr->bot; r++) // Copy in the image
        for (c=chr->curve1[r]; c < chr->curve2[r]; c++)
          if (line.pixel[r][c] & 1)
            image.pixel[r-chr->top][c-chr->left] = 255;

#ifdef VISIBLE
      xs_clr_win();
#endif
      // Now process the image
      mcursor_X = 10;  // The screen coordinates
      mcursor_Y = 10;
      cursor1_X = 10;  // The screen coordinates
      cursor1_Y = 10;
      cursor2_X = 10;  // The screen coordinates
      cursor2_Y = 10;
      cursor_col_left = cursor1_X;
      cursor_Y = 3*(image.size8 + 2));

      // Insert the left and right breaks based upon curve1 and curve2

      Breaklist_initialize(&blist, MAX_BREAKS, MAX_BRANCH, image.size8);
      dummy = Breaklist_add_break(&blist, 0, BREAK_START_OF_IMAGE, 0);
      dummy = Breaklist_add_break(&blist, image.size8*image.size8, BREAK_END_OF_IMAGE
, 0);
#ifdef VISIBLE
      print_image_with_breaks(image, &blist, 0, blist.blist_count-1, mcursor_X, mcurs
or_Y); // Print segmentation
      mcursor_Y += image.size8 + 10;
#endif

      master_segmentation(image, &blist);      // Do the segmentation

      printf("\nFinal Recognition: ");
      for (i=0; i<blist.blist_count-1; i++) {
        brk = Breaklist_return_break_ptr(&blist, i);
        if (brk->next_recog == 1)
          printf("Y ");
        else
          printf("N ");
      }
      printf("\n");
#ifdef VISIBLE
      print_image_with_breaks(image, &blist, 0, blist.blist_count-1, mcursor_X, mcurs
or_Y); // Print segmentation
      mcursor_Y += image.size8 + 10;
      xs_flush();
#endif

      printf("Final goodness segmentation.  Enter 0 to continue? ");
      j = auto_input_int();

      // Convert the breaklist and incorporate into a charlist
      breaklist_to_charlist(line, &blist, clist, &chr, chr->left);

      Breaklist_terminate(&blist);
      chr = chr->next;              // Get next character
    }
  }
```

[図181]

```
//
// Filename: multilevel2.C
// This routine contains optimized (12/91) layer 4 segmentation
//
////////////////////////////////////////////////////////////
//

//#define PASS_TO_TOSHI         // Call Toshi's routines (but ignore output)
#define DISABLE_PVVP_CACHE      // Disable the PVVP Cache

#include"cimages.h"
#include"xs.h"
#include"mymalloc.h"
#include"charlist.h"
#include"autoinput.h"
#include"math.h"

#define MAX_CHAR_HEIGHT 200  // The maximum allowed line height

////////////////////////////////
// External Variables
////////////////////////////////
extern int WINDOW_SIZE_X;
extern int WINDOW_SIZE_Y;
extern int MAX_CHAR_LEN;        // The maximum character length in pixels

////////////////////////////////
// Global Variables
////////////////////////////////

#ifdef PASS_TO_TOSHI
extern "C" toshi_recognize(int r, int c, char** image);
#endif

// The rotation table variables
#define ROT_MAX_RANGE 35        // The maximum degree range (inclusive)
#define ROT_DEG_RESOLUTION 3    // The distance in degrees
#define ROT_INDEX_RANGE 11      // = (ROT_MAX_RANGE/ROT_DEG_RESOLUTION)
int **Rot;                      // The rotated projection array:
                                //  Rot[d][h] = How many pixels shifted over when
                                //  height h and degree = (d-1)*(ROT_DEG_RESOLUTION)

#define LR_IGNORE_HEIGHT 0.25   // Minimum valley height to clip left and
                                // Edges after taking rotated projection (so
                                // minimum values on ends will be ignored

// The projection array (allocated once per line; a lot of routines use it)
int *Proj;              // Allocates memory for projection
int *VPP;               // Allocated memory for the Vertical projection

// The constants loaded from constants.dat
int max_thresh_level;   // The size of Const1 & Const2
float *Const1;          // The maximum valley depth ratio
float *Const2;          // The minimum peak height ratio

// The PVVP Cache variables
#define PVVP_CACHE_SIZE 8       // The size
int PVVP_Zero;          // -1 if the cache is empty (set this to clear!)
int PVVP_S;             // The 1st element in the cache
int PVVP_E;             // one past the last element
int PVVP_p1[PVVP_CACHE_SIZE];           // Array of p1's
int PVVP_v1[PVVP_CACHE_SIZE];           // Array of v1's
int PVVP_v2[PVVP_CACHE_SIZE];           // Array of v2's
int PVVP_p2[PVVP_CACHE_SIZE];           // Array of p2's
int PVVP_loc[PVVP_CACHE_SIZE];          // Locations of cuts
int PVVP_angle[PVVP_CACHE_SIZE];        // Angle of the cut

// Global Cursor (for plotting) variables
int Cursor1_X, Cursor1_Y;       // Line display
int Cursor2_X, Cursor2_Y;       // The final output display
int Cursor3_X, Cursor3_Y;       // The top of the columns going across
int Cursor4_X, Cursor4_Y;       // The columns going down
int Cursor_vpp_tickX, Cursor_vpp_tickY; // Where to plot lines under histogram

//************ Temp Test Routines ********************
// Print Charlist, in ascii format
extern "C" void print_asc_charlist(
  struct Charlist *clist,
  struct Character *ch)
{
  struct Character *chon;
  int i;

  chon = clist->first;
  while(chon)
  {
    if (chon == ch) printf("(");
    printf("%d-%d", chon->left, chon->right);
    if (chon == ch) printf(")");
//    printf("\n  curve1:");
//    for (i=chon->top; i<chon->bot; i++)
//      printf("%d ",chon->curve1[i]);
//    printf("\n  curve2:");
//    for (i=chon->top; i<chon->bot; i++)
//      printf("%d ",chon->curve2[i]);
    if (chon->rec == PASSED_RECOGNITION)
      printf(" Y");
    if (chon->rec == FAILED_RECOGNITION)
      printf(" N(lev=%d)",chon->level_cut_at);
    if (chon->rec == UNRECOGNIZED)
      printf(" U");
    printf("\n");

    if (chon->next != 0)
      if (chon->next->prev != chon)
      {
        printf("\n*** Charlist link error: ch->next->prev != ch !\n");
        exit(1);
      }
    chon = chon->next;

  }
  printf("\n");
}

//*************** Simple Utility Functions ***************
// Return absolute value of an int
iabs(int i)
{
  return ((i < 0) ? -i : i);
}


//*************** Initialization Routines ***************
////////////////////////////////////////////////////////////
// Initialize Const1 and Const2 (read from constants.dat)
void l4_initialize_constants()
{
  int i;
  float *a;             // Temporary storage
  float *b;             // Temporary storage
  FILE *fp;
```

【図182】

```
if ((fp = fopen("constants.dat", "r")) == NULL)
{
    cerr << "\nError in opening constants.dat\n";
    exit(1);
}

// --- This should change before the final version. When we know the size
//     of constants.dat; hardcode it!
max_thresh_level = 0;  // Stick the constants somewhere!
a = fvector(1200);
b = fvector(1200);

for (i=0; i<200; i++)
{
    fscanf(fp, "%f%f", &(a[i]), &(b[i]));
    if ((a[i] < 0) || (b[i] < 0))
    {
        max_thresh_level = i;
        break;
    }
}

if (i>=200)
{
    printf("---- Error: Only 200 entries in constants.dat is allowed!  Change the code
\n");
    exit(1);
}

const1 = fvector(max_thresh_level);
const2 = fvector(max_thresh_level);
for (i=0; i<max_thresh_level; i++)
{
    const1[i] = a[i];
    const2[i] = b[i];
}

free_fvector(a);
free_fvector(b);
}

////////////////////////////////////////////////////////////////////////
// Initialize rotation data
// (this must be called before calling compute_rotated_projection
//
void id_initialize_rot()
{
    int d_index;        // Index of degree we're working on
    double d_r_d;       // The number of degrees, and tangent term
    int r;              // Looping vars: row an. and degree on
    int d_b;

    a = ROT_INDEX_RANGE;
    b = MAX_CHAR_HEIGHT;
    Rot = imatrix(a, b);

    for (d = ROT_DEG_RESOLUTION; d<=ROT_MAX_RANGE; d+=ROT_DEG_RESOLUTION)
    {
        d_index = ((int) (d/ROT_DEG_RESOLUTION+.5)) - 1;
        d_r = tan(d * 3.14159/180.0);
        for (r=0; r< MAX_CHAR_HEIGHT; r++)
            Rot[d_index][r] = (int) (r_d*((double) r)+.5);
    }
}

//////////////////////////////////// PLOTTING ROUTINES ////////////////////////////////
void id_plot_char(
    cimage& image,      // The image
    struct Character *ch,  // The character to plot
    int x,              // The upper left hand corner (.left, .top) to plot
    int y)
{
    int r,c;
    for (r=ch->top; r < ch->bot; r++)
    {
        for (c = ch->curve1[r]; c< ch->curve2[r]; c++)
            if (image.pixel[r][c] & 1)
                xs_plotxy(x+c-ch->left, y+r);
//          xs_plotxy(x+ch->curve1[r]-2-ch->left, y+r);
//          xs_plotxy(x+ch->curve2[r]+1-ch->left, y+r);
        }
        xs_flush();
    }
}

void id_plot_charlist(
    cimage& lim,        // The image
    struct Charlist *clist,  // The clist
    int x,              // The position
    int y)
{
    struct Character *ch;
    int x_start;

    x_start = x;
    ch = clist->first;
    while(ch != 0)
    {
        if (x + ch->right - ch->left > WINDOW_SIZE_X - 20) // Wrap?
        {
            x = x_start;
            y += line.sizeX + 15;
        }
        id_plot_char(lim, ch, x, y);
        x += ch->right - ch->left + 15;
        ch = ch->next;   // Get next character
    }
}

// This routine will plot a character in the lower left hand corner
// (for the purpose of displaying to prompt for recognition)
void id_plot_lower_left(
    cimage& image,      // The image
    struct Character *ch)
{
    int r,c;
    int delta;
    delta = ch->bot - ch->top;

    for (c=0; c<ch->right-ch->left; c++)        // Erase area
        for (r=WINDOW_SIZE_Y-delta-1; r < WINDOW_SIZE_Y; r++)
            xs_unplotxy(c,r);

    for (r=ch->top; r < ch->bot; r++)           // Plot char
        for (c=ch->curve1[r]; c < ch->curve2[r]; c++)
            if (!image.pixel[r][c] & 1)
                xs_plotxy(1+c-ch->left, WINDOW_SIZE_Y-2-delta+r-ch->top);

    for (c=0; c<=ch->right-ch->left; c++)       // Plot box around it
```

```
  {
    xs_plotxy(c,WINDOW_SIZE_Y-1);
    xs_plotxy(c,WINDOW_SIZE_Y-delta-4);
  }
  for (r=WINDOW_SIZE_Y-delta-4; r < WINDOW_SIZE_Y; r++)
  {
    xs_plotxy(0, r);
    xs_plotxy(3+ch->right-ch->left, r);
  }
  xs_flush();
}


// This routine will plot a range of characters in the lower left hand corner
// (for the purposes of displaying to prompt for recognition)
// Note: They MUST have the same height range!!!
void 14_plot_lower_left_2chars(
  cimage& image,                // The image
  struct Character *ch1,        // Left character
  struct Character *ch2)        // Right character
{
  int r,c;
  int delta;

  if ((ch1->top != ch2->top) || (ch1->bot != ch2->bot))
  {
    printf("**** ERROR: 14_plot_lower_left_2chars() Called with two\n");
    printf("top's and bottoms:!!!! ch1(t,b) = (%d,%d), ch2 = (%d,%d)\n",
           ch1->top, ch1->bot, ch1->top, ch2->bot);
    exit(1);
  }
  delta = ch1->bot - ch1->top;
  for (c=0; c<15+ch1->right-ch1->left; c++)              // Erase area
    for (r=WINDOW_SIZE_Y-delta-15; r < WINDOW_SIZE_Y; r++)
      xs_unplotxy(c,r);
  for (r=ch1->top; r < ch1->bot; r++)                   // Plot char
    for (c=ch1->curve1[r]; c < ch2->curve2[r]; c++)
      if (image.pixel[r][c] & 1)
        xs_plotxy(3+c-ch1->left, WINDOW_SIZE_Y-2-delta+r-ch1->top);
  for (c=0; c<=ch2->right-ch1->left; c++)               // Plot box around it
  {
    xs_plotxy(c,WINDOW_SIZE_Y-1);
    xs_plotxy(c,WINDOW_SIZE_Y-delta-4);
  }
  for (r=WINDOW_SIZE_Y-delta-4; r < WINDOW_SIZE_Y; r++)
  {
    xs_plotxy(0, r);
    xs_plotxy(1+ch2->right-ch1->left, r);
  }
  xs_flush();
}


// This routine will start a new column on the display
void 14_plot_new_column(
  int n)                // n == 0 for double line, == 1 for single line
{
  int r;
  Cursor3_X += 20;
  Cursor4_X = Cursor3_X;
  Cursor4_Y = Cursor3_Y;
  if (Cursor4_X > (WINDOW_SIZE_X - 100))
  {
    printf("Going to clear window, Enter 0 to continue? ");
    scanf("%d", &r);
    xs_clr_win();
    Cursor3_X = Cursor4_X = 20;
  }
  if (n)
    for (r=Cursor3_Y; r<WINDOW_SIZE_Y; r++)
      xs_plotxy(Cursor3_X-10,r);
  else
    for (r=Cursor3_Y; r<WINDOW_SIZE_Y; r++)
    {
      xs_plotxy(Cursor3_X-11,r);
      xs_plotxy(Cursor3_X-9,r);
    }
  Cursor3_X += 40;              // Minimum of 40 columns over
  xs_flush();
}


//***************** TEMP RECOGNITION ROUTINE ***************
// Recognized character defined by from ch1 to ch2 (inclusive)
int recognition(                // Returns 0 for unrecognized, 1 otherwise
  cimage& image,                // The image
  struct Character *ch1,        // Left character
  struct Character *ch2)        // Right character
{
#ifdef VISIBLE
  14_plot_lower_left_2chars(image, ch1, ch2);
#endif

#ifdef PASS_TO_TOSHI
  int size_r, size_c;
  int r,c;
  char **p;
  size_r = ch1->bot - ch1->top;
  size_c = ch2->right - ch1->left;
  p = cmatrix(size_r, size_c);
  for (r=0; r<size_r; r++)
    for (c=0; c<size_c; c++)
      p[r][c] = 0;
  for (r=ch1->top; r<ch1->bot; r++)
    for (c=ch1->curve1[r]; c<ch2->curve2[r]; c++)
      p[r-ch1->top][c-ch1->left] = image.pixel[r][c];
  toshi_recognise(size_r, size_c, p);
  free_cmatrix(p, size_r);
#endif

  if ((ch2->right - ch1->left) > MAX_CHAR_LEN)
  {
    printf("Character unrecognized because it was too long\n");
    return 0;
  }
  printf("Recognized (1-yes, 0-no)? ");
  return auto_input_int();
}

////////////////////////////////////////////////////////////////////////
// This routine will determine if a character is too long (and thus exit)
int 14_too_long(        // Return 1 if too long, 0 otherwise
  struct Character *ch1,        // Left character
  struct Character *ch2)        // Right character
{
  if ((ch2->right - ch1->left) > MAX_CHAR_LEN)
    return 1;
  return 0;
}


//*************** Character Utility Routines ********************
// This routine will move ch->left, and ch->right so they they are
//   set to the 1st and last 'on' pixel in the image.  This is used to
```

```
//   eliminate white space from the left and right side of a character
//   after it is cut so the size won't be too inaccurate.
//   Call when place the cut, ch1 = left char, ch2 = right char
void l4_char_reset_lr(
    cimage& image,
    struct Character *ch1,      // The left character (only does right side)
    struct Character *ch2)      // The right character (only does left side)
{
  int r,c;
  int new_left, new_right;      // The new edges of the characters

  // Right side of char1: Look for 1st on pixel
  for (c = ch1->right-1; c>=ch1->left; c--)
    for (r = ch1->top; r<ch1->bot; r++)
      if ((c >= ch1->curve1[r]) && (c < ch1->curve2[r]))
        if (image.pixel[r][c] & 1)      // If pixel is on
        {
          new_right = c+1;
          goto exit_loop;                       // Break out of the loop
        }
  exit_loop:

  // Set right side of character to this value
  ch1->right = new_right;
  for (r=ch1->top; r<ch1->bot; r++)
    if (ch1->curve1[r] > new_right)
    {
      ch1->curve1[r] = new_right;
      if (ch1->curve1[r] > new_right)
        ch1->curve2[r] = new_right;
    }

  // Do the left side of ch2
  // Look for the 1st on pixel
  for (c=ch2->left; (c<ch2->right); c++)
    for (r=ch2->top; r < ch2->bot; r++)
      if ((c >= ch2->curve1[r]) && (c < ch2->curve2[r]))
        if ((image.pixel[r][c] & 1) == 1)
        {
          new_left = c;
          goto exit_loop2;              // Break out of the loop
        }
  exit_loop2:

  // Set the left side of character to this value
  ch2->left = new_left;
  for (r=ch2->top; r<ch2->bot; r++)
    if (ch2->curve1[r] < new_left)
    {
      ch2->curve1[r] = new_left;
      if (ch2->curve2[r] < new_left)
        ch2->curve2[r] = new_left;
    }
}


//*************** THE ROTATED PROJECTION ROUTINES ***************
void l4_find_projection(
    cimage& image,      // The input image
    struct Character *ch, // The character to find the projection on
    int angle,          // The rotation/ROT_DEG_RESOLUTION: Must be withing a
                        //   valid range!
    int range1,         // The column range to search through
    int range2,         //
    int *ang_proj,      // The matrix to put the projection into (*MUST*
                        //   be previously allocated!)
    int *ang_proj_min,  // The min index of ang_proj[] (inclusive) (output)
    int *ang_proj_max)  // The max index of ang_proj[] (exclusive) (output)
                        // Note: Projection put in Proj[]
{
  int r,c;              // Looping variables

  // Do a standard histogram
  if (angle == 0)       // Do a standard vpp histogram
  {
    *ang_proj_min = range1;
    if (*ang_proj_min < ch->left)
      *ang_proj_min = ch->left;
    *ang_proj_max = range2;
    if (*ang_proj_max > ch->right)
      *ang_proj_max = ch->right;

    for (c=ch->left; c<ch->right; c++)  // Zero projection array
      ang_proj[c] = 0;
    for (r=ch->top; r<ch->bot; r++)     // Compute the projection
      for (c=ch->curve1[r]; c<ch->curve2[r]; c++)
        if (image.pixel[r][c] & 1)
          (ang_proj[c])++;
    return;
  }

  // Do a theta is greater than zero
  if (angle > 0) {
    angle--;                    // Get the index to use int Rot
    *ang_proj_min = range1;
    if (*ang_proj_min < ch->left)
      *ang_proj_min = ch->left;
    *ang_proj_max = range2;
    if (*ang_proj_max > ch->right)
      *ang_proj_max = ch->right;
    *ang_proj_min -= Rot[angle][ch->bot-ch->top];

    for (c=*ang_proj_min; c<*ang_proj_max; c++) // Zero projection array
      ang_proj[c] = 0;
          for (r=ch->top; r<ch->bot; r++)       // Compute the projection
      for (c=ch->curve1[r]; c<ch->curve2[r]; c++)
        if (image.pixel[r][c] & 1)
          (ang_proj[c-Rot[angle][ch->bot-r-1]])++;
    return;
  }

  // Do a theta is less than zero
  angle = -angle - 1;           // Get the index to use in Rot
  *ang_proj_min = range1;
  if (*ang_proj_min < ch->left)
    *ang_proj_min = ch->left;
  *ang_proj_max = range2;
  if (*ang_proj_max > ch->right)
    *ang_proj_max = ch->right;
  *ang_proj_max += Rot[angle][ch->bot-ch->top];

  for (c=*ang_proj_min; c<*ang_proj_max; c++)   // Zero projection array
    ang_proj[c] = 0;
  for (r=ch->top; r<ch->bot; r++)       // Compute the projection
    for (c=ch->curve1[r]; c<ch->curve2[r]; c++)
      if (image.pixel[r][c] & 1)
        (ang_proj[c+Rot[angle][ch->bot-r-1]])++;

  return;
}
```

[図184]

【図185】

```
//.....The PVP cache routines.....
// This routine will insert an entry into the PVP cache (A cyclic cache)
// with inputs p1,v1,p2 and output loc,angle.
void id_PVP_cache_insert(
    int p1,     // The top of 1st peak
    int v1,     // The start of valley
    int v2,     // The end of valley
    int p2,     // The top of 2nd peak
    struct Character *ch, // The character we're in
    int min_theta,  // The final angle
    int min_at)     // Where the minimum is at
{
#ifdef DISABLE_PVP_CACHE
  return;
#endif

  int r;

  // Make sure the PVP area doesn't have a cut running through it:
  for (r=ch->top; r < ch->bot; r++)
    if ((ch->curve2[r] > p1) || (ch->curve2[r] < p2))
      return;   // If so, don't put on PVP Que

  if (PVP_Zero)
  {
    PVP_S = 0;      // Insert 1st element
    PVP_E = 1;
    PVP_Zero = 0;
  } else {
    if (PVP_S == PVP_E)     // List full, swap last out
    {
      // The next line is basically PVP_S = (PVP_S-1) & CACHE_SIZE
      PVP_S = (PVP_S==0) ? PVP_S=PVP_CACHE_SIZE-1 : PVP_S-1;
      PVP_E = PVP_S;
    } else {
      PVP_S = (PVP_S==0) ? PVP_S:PVP_CACHE_SIZE-1 : PVP_S-1;
    }
  }
  // Now place the new data at PVP_S:
  PVP_p1[PVP_S] = p1;
  PVP_v1[PVP_S] = v1;
  PVP_v2[PVP_S] = v2;
  PVP_p2[PVP_S] = p2;
  PVP_loc[PVP_S] = min_at;
  PVP_angle[PVP_S] = min_theta;
}

// This routine will check PVP against past results and if we've done
// it before, it will just grab the results.  Return 0 if not hit, 1 if hit
int id_PVP_cache_hit(  // Return 0 if no hit
    int p1,     // The top of 1st peak
    int v1,     // The start of valley
    int v2,     // The end of valley
    int p2,     // The top of 2nd peak
    struct Character *ch, // The character we're in
    int *min_theta,  // Output: The final angle
    int *min_at)     // Output: Where the minimum is at
{
  int i,r;

#ifdef DISABLE_PVP_CACHE
  return 0;
#endif

  if (PVP_Zero)     // Return 0 if miss
    return 0;

  i = PVP_S;
  do
  {
    if ((PVP_p1[i] == p1) && (PVP_v1[i] == v1) &&    // If hit!
        (PVP_v2[i] == v2) && (PVP_p2[i] == p2))
    {
      // Ensure the PVP area doesn't have a cut running through it:
      for (r=ch->top; r < ch->bot; r++)
        if ((ch->curve2[r] > p1) || (ch->curve2[r] < p2))
          return 0;     // If so, return a miss
      *min_theta = PVP_angle[i];
      *min_at = PVP_loc[i];
      return 1;
    }
    i++;
    if (i >= PVP_CACHE_SIZE)
      i = 0;
  } while (i != PVP_E);     // Loop until it hits the end
  return 0;     // None found
}

//.....
//..... The find peak and valley routines
//////////////////////////////////////////////////////////////////////
// Subroutine to find next peak to the right in a projection
// -1 - none found
int find_peak_to_right(
    int proj[],     // The projection (an array)
    int proj_size,  // The max size of the projection (not inclusive)
    int pos,        // Where to start looking
    int min_peak_height)    // Minimum value for the peak
{
  int peak;     // Where the peak is

  peak = -1;    // Set to not found
  while ((++pos < proj_size) && (peak == -1))
    if ((proj[pos-1] >= proj[pos]) && (proj[pos-1] >= min_peak_height))
      peak = pos - 1;
  return peak;
}

//////////////////////////////////////////////////////////////////////
// Subroutine to find next peak to the left in a projection
// -1 - none found
int find_peak_to_left(
    int proj[],     // The projection
    int proj_min,   // The leftmost index value of proj[]
    int pos,        // Where to start looking
    int min_peak_height)    // Minimum value for the peak
{
  int peak;     // Where the peak is

  peak = -1;    // Set to not found
  while ((--pos >= proj_min) && (peak == -1))
    if ((proj[pos-1] >= proj[pos]) && (proj[pos-1] >= min_peak_height))
      peak = pos - 1;
  return peak;
}

//////////////////////////////////////////////////////////////////////
// Subroutine to find next valley to the right in a projection
// -1 - none found
int find_valley_to_right(
```

【図186】

```
  int proj[],           // The projection
  int proj_size,        // The max size of the projection (not inclusive)
  int pos,              // Where to start looking
  int max_valley_depth) // Highest point a valley can be
{
  int valley;           // Where the valley is

  valley = -1;          // Set to not found
  while ((++pos < proj_size) && (valley == -1))
    if ((proj[pos-1] <= proj[pos]) && (proj[pos-1] <= max_valley_depth))
      valley = pos - 1;
  return valley;
}


////////////////////////////////////////////////////////////////////////
// Subroutine to find next valley to the left in a projection
// -1 = none found
int find_valley_to_left(
  int proj[],           // The projection
  int proj_min,         // The leftmost index value of proj[]
  int pos,              // Where to start looking
  int max_valley_depth) // Maxiaim value for the valley
{
  int valley;           // Where the peak is

  valley = -1;          // Set to not found
  while ((--pos >= proj_min) && (valley == -1))
    if ((proj[pos+1] <= proj[pos]) && (proj[pos+1] <= max_valley_depth))
      valley = pos + 1;
  return valley;
}


//*************** Find the angles of the valley ***************
// This routine will find the locations of the Peak-Valley-Peak (the
// P-V-V-P Coords) of the histogram in VPP[]
void l4_find_PVVP(
    int VPP_min,                // The min range of the VPP
    int VPP_max,                // The max range of the VPP
    int valley_pos,             // The position to the right if the 1st peak
    int min_peak_height,        // Constants
    int max_valley_depth,       // The max valley depth
    int *p1,                    // The top of 1st peak
    int *v1,                    // The start of valley
    int *v2,                    // The end of valley
    int *p2)                    // The top of 2nd peak
{
  // Find p1: the peak to the left of the valley
  *p1 = find_peak_to_left(VPP, VPP_min, valley_pos, min_peak_height);
  if (*p1 == -1)            // If none found, make it the left side
    *p1 = VPP_min;

  // Find v1: the valley to the right of the given peak
  *v1 = find_valley_to_right(VPP, VPP_max, *p1, max_valley_depth);
  if (*v1 == -1)            // If not found, make it the right side
    *v1 = VPP_max - 1;

  // Find p2: the next peak to the right of valley v1
  *p2 = find_peak_to_right(VPP, VPP_max, *v1, min_peak_height);
  if (*p2 == -1)            // If not found, make it the right side
    *p2 = VPP_max - 1;

  // Find v2: the nearest valley to the left of peak at p2
  *v2 = find_valley_to_left(VPP, VPP_min, *p2, max_valley_depth);
  if (*v2 == -1)            // If not found, make it the left side
    *v2 = VPP_min;
}


////////////////////////////////////////////////////////////////////////
// This routine will find the angles given the P-V-V-P Coordinates
// (The array is in VPP)
void l4_find_valley_angles(
    int p1,                     // The top of 1st peak
    int v1,                     // The start of valley
    int v2,                     // The end of valley
    int p2,                     // The top of 2nd peak
    int *theta1,                // Output: The index of angles
    int *theta2)                // ( = angles/ROT_DEG_RESOLUTION)

{
  int p1y;              // The height of left peak
  int v1y;              // The height of left valley bottom
  int v2y;              // The height of right valley
  int p2y;              // The height of right peak
  double t1, t2;        // The output angle in radians

  p1y = VPP[p1];        // Get its height
  v1y = VPP[v1];
  v2y = VPP[v2];
  p2y = VPP[p2];

  t1 = atan((double) (v1-p1)/ (double) (v1y-p1y)); // This is positive
  t2 = atan((double) (p2-v2)/ (double) (p2y-v2y)); // This is negative

  t1 = t1 * 180.0 / 3.14159 / ROT_DEG_RESOLUTION;
  t2 = t2 * 180.0 / 3.14159 / ROT_DEG_RESOLUTION;

  *theta1 = (int) (t1+.5);      // Round the angles
  *theta2 = (int) (t2-.5);
}

// *************** Find minimum in a rotated valley  **************
void l4_find_min_valley_at_theta(
    cimage& image,              // The image
    struct Character *ch,       // The character to find the projection on
    int angle,                  // The angle (scaled)
    int p1,                     // The left range to look through
    int p2,                     // The right range to look through
    int *min,                   // The best minimum found
    int *min_at,                // It's location
    int *min_theta)             // The angle where it's located
{
  int pmin, pmax;               // Min and max indices of Proj
  int max_value_proj;           // The maximum value of the projection
  int r1,r2;                    // Ranges to search for minimum (inclusive!)
  int lr_height;                // The minimum height needed for a peak (so
                                //    left & right sides won't have a minimum)
  int i;

  if (*min == 0)                // If a zero found!
    return;                     // Don't bother doing the rest!

  // Find the projection (load into Proj[])
  l4_find_projection(image, ch, angle, p1, p2, Proj, &pmin, &pmax);

  max_value_proj = Proj[pmin];
  for (i=pmin+1; i<=pmax; i++)
    if (Proj[i] > max_value_proj)
      max_value_proj = Proj[i];

  // Choose range to ignore points before first peaks on left and right
  r1 = pmin;
```

【図187】

```
    r2 = pmax-1;
    lr_height = (int) ((float) max_value_proj * LR_IGNORE_HEIGHT+.5);
                                    // How high to ignore the ends

    // Find left peak (while slope is still rising), and above LR_IGNORE_HEIGHT
    while ((Proj[r1] <= Proj[r1-1]) || (Proj[r1-1] <= Proj[r1-2]) ||
           (Proj[r1-1] <= Proj[r1-3]) || (Proj[r1-2] < lr_height))
      if ((--r1) >= pmax - 3)       // If got to left edge
        return;

    // Find right peak (while slope is rising (from right to left))
    while ((Proj[r2] <= Proj[r2+1]) || (Proj[r2+1] <= Proj[r2+2]) ||
           (Proj[r2] <= Proj[r2+3]) || (Proj[r2] < lr_height))
    {
      r2--;
      if ((r2 <= r1) || (r2 <=pmin-3))    // If reached the left point
        return;
    }

    // Search this range, and find best minimum
    for (i=r1; i<=r2; i++)
    {
      if (Proj[i] <= *min)          // Found a better minimum
      {
        if (Proj[i] != *min)
        {
          *min = Proj[i];
          *min_at = i;
          *min_theta = angle;
        } else {
          if (labs(angle) < labs(*min_theta))
          {
            printf("Tiebreaker broken.....\n");
            *min_at = i;
            *min_theta = angle;
          }
        }
      }
    }


    /*********** The find optimal rotated histogram routines  **********
    ////////////////////////////////////////////////////////////////////
    // The insert angular break: Attempts to insert break in a character at a given
    // thresh level.  This updates ch->rec and ch->level_cut_at for both
    // characters
    int insert_angular_break(       // Returns number chars (1 or 0)
      CHARLIST *charlist,           // The character list to add chars to
      struct Charlist *ch,          // The character to be cut
      int thresh_level,             // The threshold levels
      int repeating_char)           // = 1 if the 1st time this routine was called
                                    // set on the same character and no breaks
                                    // were inserted (so we don't have to recompute
                                    // the VPP, etc)
    {
      static int VPP_min, VPP_max;  // The start and stop values of the VPP
      static int max_value_VPP;     // The maximum value of the VPP
      int max_valley_depth;         // Maximum height for a valley
      int min_peak_height;          // Minimum height for a peak
      int valley_pos;               // Position of valley
      int angle, angle2;            // Angles of valley walls (/ROT_DEG_RESOLUTION)
      int left_r2,right_r2;         // The Peak-Valley-Peak column numbers
      int min_at;                   // The location of the minimum
      int min_theta;                // The angle of the minimum
      int min;                      // The value of the minimum
      int range_r1_min;             // Range of angles to search for in theta
      int range_r1_max;             //
      int range_r2_min;             // 2nd range of angles (ditto)
      int range_r2_max;             //
      int angle;                    // What angle we're finding the projection for
      struct Charlist *newchar;     // The new character to be inserted
      int i,j,k,r,t;

      if (repeating_char == 0)      // Remember last if recomputing
      {
        id_find_projection(image, ch, 0, ch->left, ch->right, &VPP_min, &VPP_max);

        // Find maximum projection (for find peak and valley heights)
        max_value_VPP = VPP[VPP_min];
        for (i=VPP_min+1; i<=VPP_max; i++)
          if (VPP[i] > max_value_VPP)
            max_value_VPP = VPP[i];
      } else {
        printf("-- VPP Remembered: Max_value_VPP = %d\n", max_value_VPP);
      }

      // Compute max_valley_depth and min_peak_height
      max_valley_depth = (int) ((float) max_value_VPP * Const1[thresh_level]);
      min_peak_height = (int) ((float) max_value_VPP * Const2[thresh_level]);
      if (max_valley_depth == min_peak_height)
        min_peak_height++;

      printf("max_value_VPP = %d\n", max_value_VPP);
      printf("Threshlevel = %d\n", thresh_level);
      printf("Const1,2 = %f, %f\n", Const1[thresh_level], Const2[thresh_level]);
      printf("max_valley_depth = %d\n", max_valley_depth);
      printf("min_peak_height = %d\n", min_peak_height);

      /*** BEGIN PLOTTING ***/
    #ifdef VISIBLE
      int c;
      if (repeating_char == 0)
      {
        id_plot_new_column(0);
        id_plot_char(image, ch, Cursor_X, Cursor_Y);    // Plot character
        Cursor_Y += ch->box.h + 4;
        Cursor_X += ch->box.w;
        for (r=VPP_min;r<=VPP_max;r++)                  // Plot VPP
          for (k=0; k<VPP[r]; k++)
            xs_plotxy(Cursor_X + r - VPP_min, Cursor_Y - k);
        if (Cursor_X < Cursor_X + VPP_max - VPP_min)    // Adjust next col loc
          Cursor_X = Cursor_X + VPP_max - VPP_min;
        xs_flush();
        Cursor_VPP_pickX = Cursor_X;
        Cursor_VPP_pickY = Cursor_Y + 3;
        Cursor_Y += 20;
      }
    #endif
      /*** END PLOTTING ***/

      // Find first peak to right
      i = find_peak_to_right(VPP, VPP_max, VPP_min, min_peak_height);
      if (i == -1)                  // If peak not found...
        return 0;

      // Search for valley to right
      // Note: i should be the peak on the left!
      valley_pos = find_valley_to_right(VPP, VPP_max, i, max_valley_depth);
      if (valley_pos == -1)         // If not found
        return 0;
```

【図１８８】

```
// Search for next peak to right of valley_pos
i2 = find_peak_to_right(VPP, VPP_max, valley_pos, min_peak_height,
                        // If not found
if (i2 == -1)
return 0;

// Now compute the angles
i4_find_PVVP(VPP_min, VPP_max, valley_pos, min_peak_height,
             max_valley_depth, &p1, &v1, &v2, &p2);

/*** BEGIN PLOTTING ***/
#ifdef VISIBLE
// Plot the tick mark under valley found
for (c = p1; c <= p2; c++)
  xs_plotxy(Cursor_VPP_tick+c-VPP_min, Cursor_VPP_tickY-1);
xs_plotxy(Cursor_VPP_tick+p1-VPP_min, Cursor_VPP_tickY-1);
xs_plotxy(Cursor_VPP_tick+p2-VPP_min, Cursor_VPP_tickY-1);
xs_plotxy(Cursor_VPP_tick+v1-VPP_min, Cursor_VPP_tickY-1);
xs_plotxy(Cursor_VPP_tick+v2-VPP_min, Cursor_VPP_tickY-1);
xs_flush();
#endif
/*** END PLOTTING ***/

// If we've done this VPP before, get results.  If not, compute them
if (i4_PVVP_cache_hit(p1,v1,v2,p2,ch,&min_theta,&min_at) == 0)
{
  i4_find_valley_angles(p1, v1, v2, p2, &theta1, &theta2);
  printf("Valley Angles = %d, %d\n", theta1, theta2);
//

  range_t1_min = theta1-3;   // theta1 - 3.5
  range_t1_max = theta1+3;   // theta1 + 3.5
  range_t2_min = theta2-3;   // theta2 - 3.5
  range_t2_max = theta2+3;   // theta2 + 3.5

  // Check ranges
  if (range_t1_max > NOT_INDEX_RANGE)
    range_t1_max = NOT_INDEX_RANGE;
  if (range_t2_max > NOT_INDEX_RANGE)
    range_t2_max = NOT_INDEX_RANGE;
  if (range_t1_min < -NOT_INDEX_RANGE)
    range_t1_min = -NOT_INDEX_RANGE;
  if (range_t2_min < -NOT_INDEX_RANGE)
    range_t2_min = -NOT_INDEX_RANGE;

  if (range_t1_max >= range_t1_min)  // If overlap :(
  {
    printf("* Range reduced; (%d:%d), (%d:%d) to ", range_t1_min,
      range_t1_max, range_t2_min, range_t2_max);
    // Then combine & do one search
    range_t1_max = range_t2_max;
    range_t1_min = range_t2_max + 1;
    printf("(%d:%d)\n", range_t1_min, range_t1_max);
  }

  // Search the angle range and find the best breakbreak the image
                        // Search 0 degrees
  min = VPP[v2];
  min_at = v2;
  min_theta = 0;
  for (j=v1; j<v2; j++)
    if (VPP[j] < min)
    {
      min = VPP[j];
      min_at = j;
    }
  // Search rest of them
  for (angle = range_t1_min; angle <= range_t1_max; angle++)
    i4_find_min_valley_at_theta(image, ch, angle, p1, p2,
                                &min, &min_at, &min_theta);

  for (angle = range_t2_min; angle <= range_t2_max; angle++)
    i4_find_min_valley_at_theta(image, ch, angle, p1, p2,
                                &min, &min_at, &min_theta);

  i4_PVVP_cache_insert(p1,v1,v2,p2,ch,min_theta,min_at);
                        // Place in cache
//  printf("Minimum found: Angle = %d, at %d = %fn", min_theta, min_at, min);
//  } else {
//    printf("*** Cache HIT!\n");
//    printf("Cache: Angle = %d, at %fn", min_theta, min_at);
//  }

/*** BEGIN PLOTTING ***/
#ifdef VISIBLE
  char string[100];
  int pmin, pmax, max_value_proj, tx, ty, tmin;
  if (Cursor_X > WINDOW_SIZE_X - 10)
                        // Wrap over to new column
    i4_plot_new_column(1);
  sprintf(string, "Lev %d", thresh_level);
  xs_string(Cursor_X, Cursor_Y, string);
                        // Plot level
  Cursor_Y += 8+3;
  sprintf(string, "Deg %d", min_theta*3);
  xs_string(Cursor_X, Cursor_Y, string);
                        // Plot angle
  Cursor_Y += 8+1;

  // Load rotated projection into array again
  i4_find_projection(image, ch, min_theta, p1, p2, Proj, &pmin, &pmax);
  max_value_proj = Proj[pmin];
                        // Compute height of Proj[]
  for (i=pmin; i<pmax; i++)
    if (Proj[i] > max_value_proj)
      max_value_proj = Proj[i];
  Cursor_Y += max_value_proj;
                        // Plot Proj[]
  for (i=pmin; i<pmax; i++)
    for (k=0; k<=Proj[i]; k++)
      xs_plotxy(Cursor_X+i-pmin, Cursor_Y-k);
  if (Cursor_X < Cursor_X + pmax - pmin)
                        // Adjust next col loc
    Cursor_X = Cursor_X + pmax - pmin;
  c = Cursor_X + min_at - pmin;
  r = Cursor_Y-2;
  xs_plotxy(c, r);                      // Plot Tickmark
  xs_plotxy(c-1, r+1); xs_plotxy(c+1, r+1);
  xs_plotxy(c-2, r+2); xs_plotxy(c+2, r+2);
  xs_plotxy(c-3, r+3); xs_plotxy(c+3, r+3);
  Cursor_Y += 10;

  tx = Cursor_X; ty = Cursor_Y;
  if (Cursor_X < Cursor_X + ch->right - ch->left)
                        // Adjust next col loc
    Cursor_X = Cursor_X + ch->right - ch->left;
  i4_plot_char(image, ch, tx, ty);
  Cursor_Y += ch->bot - ch->top + 9;

  if (min_theta > 0)                    // Plot cut line
  {
    tmin = min_theta-1;
    for (r=ch->top-6; r<ch->bot; r++)
      xs_plotxy(tx+Rot[tmin][ch->bot-r-1]+min_at-ch->left, ty+r);
  }
  if (min_theta < 0)                    // Plot cut line
  {
    tmin = -min_theta-1;
    for (r=ch->top-6; r<ch->bot; r++)
      xs_plotxy(tx-Rot[tmin][ch->bot-r-1]+min_at-ch->left, ty+r);
  }
  if (min_theta == 0)                   // Plot cut line
  {
    for (r=ch->top-6; r<ch->bot; r++)
      xs_plotxy(tx+min_at-ch->left, ty+r);
```

【図189】

```
    }

    Cursor4_Y += 20;                    // Move cursor down

    XA_flush();
#endif /*** END PLOTTING ***/


    // Break the Character
    newchar = &(clist->chr[clist->size]);  // Set newchar to point to the end
                                           // of the charlist (the next blank character)

    if (min_theta == 0)         // Horizontal break
    {
      for (r=ch->top; r<=ch->bot; r++)
      {
        newchar->curve2[r] = ch->curve2[r];      // Break the character
        b = MIN_AT;
        if (b < ch->curve1[r])
          b = ch->curve1[r];
        if (b > ch->curve2[r])
          b = ch->curve2[r];
        newchar->curve1[r] = ch->curve2[r] = b;
      }
      newchar->right = ch->right;
      ch->right = MIN_AT;
      newchar->left = MIN_AT;
    }
    else if (min_theta > 0)  // Positive angled break
    {
      t = min_theta-1;  // The index to use for the images
      for (r = ch->top; r<=ch->bot; r++)
      {
        newchar->curve1[r] = ch->curve1[r];
        b = MIN_AT + BOI[t][ch->bot-r-1];
        if (b < ch->curve1[r])
          b = ch->curve1[r];
        if (b > ch->curve2[r])
          b = ch->curve2[r];
        newchar->curve2[r] = ch->curve1[r] = b;
      }
      newchar->right = ch->right;
      ch->right = MIN_AT + BOI[t][ch->bot-1];
      newchar->left = MIN_AT;     // Negative angled break
    } else {
      t = -min_theta-1;
      for (r=ch->top; r<=ch->bot; r++)
      {
        newchar->curve1[r] = ch->curve1[r];
        b = MIN_AT - BOI[t][ch->bot-r-1];
        if (b < ch->curve1[r])
          b = ch->curve1[r];
        if (b > ch->curve2[r])
          b = ch->curve2[r];
        newchar->curve2[r] = ch->curve1[r] = b;
      }
      newchar->right = ch->right;
      ch->right = MIN_AT;
      newchar->left = MIN_AT - BOI[t][ch->bot-1];
    }
  }
  newchar->top = ch->top;
  newchar->bot = ch->bot;
  newchar->level_cut_at = thresh_level-1;
  ch->level_cut_at = thresh_level;
  newchar->src = UNRECOGNIZED;
  ch->src = UNRECOGNIZED;

  ia_char_reset_lr(image, ch, newchar); // Update left & right borders
  Charlist_insert_after_cut(clist, ch);

  return 1;
}


//.....................................................................
//.....................................................................
//.....                                                           .....
//.....                THE GOODCUT ROUTINES SECTION               .....
//.....                                                           .....
//.....................................................................
//.....................................................................

////////////////////////////////////////////////////////////////////////
// This routine will cut a character, starting at level 0, and going up
// through the levels until it makes a break.  Returns 1 if successful,
// 0 if unsuccessful.
int ia_cut_character(
  cImage image,
  struct Charlist *clist,      // The character list to add chars to
  struct Character *ch)        // The character to be cut
{
  int level;                   // The level of the break
  int repeat;                  // If repeating

  repeat = 0;
  for (level = 0; level < MAX_THRESH_LEVEL; level++)  // Loop through levels
  {
    if (insert_angular_break(image, clist, ch, level, repeat))
      return 1;
    repeat = 1;                // We're repeating from now on
  }

  return 0;                    // No break inserted.
}


////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////
void ia_combine_chars(
  struct Charlist *clist,
  struct Character *ch1,       // The character to start from
  struct Character *ch2)       // The character to start from
{
  struct Character *r;         // Looping character
  int i;
  if (ch1 == ch2)              // Return if equal
    return;
  if ((ch1->top != ch2->top) || (ch1->bot != ch2->bot))
  {
    printf("Error in ia_combine_chars(): Combing characters who's top's won't\n");
    printf("and bottoms are not equal: (%,%d), (%d,%d)\n",
      ch1->top, ch1->bot, ch2->top, ch2->bot);
    exit(1);
  }

  for (i=ch1->top; i<=ch1->bot; i++)
    ch1->curve2[i] = ch2->curve2[i];
  ch1->right = ch2->right;

  // Remove all the characters between the two
  r = ch2;
  while (r != ch1)
  {
```

[図190]

```
    Charlist_remove_char(clist, c);
    c = c->prev;
  }
  Charlist_recalculate_loc(ch1);        // Recalculate location
}


//////////////////////////////////////////////////////////////////////////
// This routine works with 14_goodcut_comb_left() to check '99''s after
// successful recognition:  Ie.  N N(99)  Y  tries N (N(99)  Y)   which
// will detect and remove some errors caused by an false successful recognition
int 14_goodcut_99_left(
    cimage& image,
    struct Charlist *clist,
    struct Character *ch,       // The character to start from
    struct Character *lch)      // Leftmost character to search (exclusive)
{
  struct Character *c;          // The leftmost character of the range
  int ret_value;                // The # of chars removed
  int temp_value;

  ret_value = temp_value = 0;
  c = ch;

  while(1)
  {
    c=c->prev;                  // Get the previous character
    temp_value++;
    if (c == lch)               // At edge
      return ret_value;
    if (c->rec != FAILED_RECOGNITION)   // Return if not a N
      return ret_value;
    if (c->level_cut_at != max_thresh_level)    // Return if not '99'
      return ret_value;
    if (recognition(image,c,ch))
    {
      14_combine_chars(clist, c,ch);    // Combine it into one character (in c)
      c->rec = PASSED_RECOGNITION;
      ret_value += temp_value;
      temp_value = 0;
      ch = c;
    }
  }
}


//////////////////////////////////////////////////////////////////////////
// This routine will start at character ch and search to left from ch
// grouping characters to see if they're recognized.  If so they are
// combined.  Note: lch can == 0.
14_goodcut_comb_left(
    cimage& image,
    struct Charlist *clist,
    struct Character *ch,       // The character to start from
    struct Character *lch)      // Leftmost character to search (exclusive)
{
  struct Character *c;          // The leftmost character of the range
  int ret_value;                // The # of chars removed

  ret_value = 0;
  c = ch;                       // Start at character we're on

  while(1)                      // This loop is 'return'ed out of
  {
    if (recognition(image, c, ch))  // c to ch recognized?
    {
      14_combine_chars(clist, c,ch);    // Combine it into one character (in c)
      c->rec = PASSED_RECOGNITION;
      ret_value += 14_goodcut_99_left(image,clist,c,lch); // Do the 99's
      return ret_value;
    }
    if (c == ch)
      c->rec = FAILED_RECOGNITION;
    c = c->prev;                // Incorporate next character to left
    ret_value++;

    // Check for exits
    if (c == lch)               // If we're at the end, exit
      return 0;
    if (c->next->rec == PASSED_RECOGNITION)     // If we've passed a Y
      return 0;                 // Stop searching
    if (14_too_long(c,ch))      // If character is too long
      return 0;
  }
}


//////////////////////////////////////////////////////////////////////////
// This routine works with 14_goodcut_comb_right() to check '99''s after
// successful recognition, or when a top level is hit:
//      Ie.  N N(99) ch  tries N (N(99) ch)   which
// will detect and remove some errors caused by an false successful recognition
// and by not checking adjacent characters when can't split at level 99
int 14_goodcut_99_right(
    cimage& image,
    struct Charlist *clist,
    struct Character *ch,       // The character to start from
    struct Character *rch)      // Rightmost character to search (exclusive)
{
  struct Character *c;          // The rightmost character of the range
  int ret_value;                // The # of chars removed
  int temp_value;

  ret_value = temp_value = 0;
  c = ch;

  while(1)
  {
    c=c->next;                  // Get the previous character
    temp_value++;
    if (c == rch)               // At edge
      return ret_value;
    if (c->rec != FAILED_RECOGNITION)   // Return if not a N
      return ret_value;
    if (c->level_cut_at != max_thresh_level)    // Return if not '99'
      return ret_value;
    if (recognition(image,ch,c))
    {
      14_combine_chars(clist, ch,c);    // Combine it into one character (in c)
      ch->rec = PASSED_RECOGNITION;
      ret_value += temp_value;
      temp_value = 0;
      c = ch;
    }
  }
}


//////////////////////////////////////////////////////////////////////////
// This routine will start at character ch and search to right from ch
// grouping characters to see if they're recognized.  If so they are
// combined.  Note: lch can == 0.
14_goodcut_comb_right(
    cimage& image,
```

【図191】

```
    struct Charlist *clist,     // The character to start from
    struct Character *ch,       // Rightmost character to search (exclusive)
    struct Character *rch)
{
  struct Character *c;          // The RIGHTMOST character of the range
  int ret_value;                // The # of chars removed

  ret_value = 1;

  c = ch;                       // Start at character before ch

  while(1)                      // This loop is *return*ed out of
  {
    if (recognition(image, ch, c))  // ch to c recognised)
    {
      14_combine_chars(clist, ch,c);  // combine it into one character (in ch)
      ch->rec = PASSED_RECOGNITION;
      ret_value += 14_goodcut_99_right(image,clist,ch,rch);  // Do the 99'sc
      return ret_value;
    }
    if (ch == c)
      ch->rec = FAILED_RECOGNITION;
                                  // Incorporate next character to right
    c = c->next;
    ret_value++;

    // Check for exits
    if (c == rch)                 // If we're at the end; exit
      return 0;
    if (c->prev->rec == PASSED_RECOGNITION)   // If we've passed a Y
                                  // Stop matching
    if (14_too_long(ch,c))        // If character is too long
      return 0;
  }
}

//.....
// The goodcut routine
//.....
// This will run goodcut on a character to be broken up
int goodcut_break_character(    // Returns number of chars added
    IMAGE *image,
    struct Charlist *clist,
    struct Character *ch)
{
  struct Character *left_border;      // The left border character
  struct Character *right_border;     // The left border character
  struct Character *cut;              // The character to cut
  struct Character *ch;               // A looping variable
  int chars_cut;                      // The number of characters added
  int lowest_level;                   // The level of the character to cut
  int a;

  chars_cut = 0;
  left_border = ch->prev;
  right_border = ch->next;

  while(1)                            // Loop until broken out of
  {
    lowest_level = max_thresh_level;  // Get 1st character
    ch = ch;
    while (ch != right_border)        // Loop until searched all characters
    {
      if (ch->rec == FAILED_RECOGNITION)      // If N
        if (ch->level_cut_at < lowest_level)  // If it was cut at a lower level
        {
          lowest_level = ch->level_cut_at;   // then cut this char next
          cut = ch;
        }
      ch = ch->next;
    }
    if (lowest_level == max_thresh_level) // If we searched all the chars
      break;                              // Exit if no character to cut
    if (14_cut_character(image,clist,cut) == 0) // ******* DELETE ME ********
      break;                              // Break the character
    else
    {
      cut->level_cut_at = max_thresh_level;   // Can't break character
      chars_cut++;                            // Try to combine the characters
      ch = cut->next;                         // Remembered in case next line removes char 'cut'
      chars_cut -= 14_goodcut_comb_left(image,clist,cut,left_border);
      chars_cut -= 14_goodcut_comb_right(image,clist,ch,right_border);
    }
  }

  return chars_cut;
}

//.....
// This passes each character to the goodcut routine
int goodcut(                    // Returns number of characters added
    IMAGE *image,
    struct Charlist *clist)
{
  int chars_added;              // The number of characters added (output)
  struct Character *ch;         // Used when looping (the beginning of the chunck)
  struct Character *next_ch;    // Temp storage character (where to continue)

  chars_added = 0;
  // Set all break levels to -1 and
  // pass all the unrecognised characters to recognition
  ch = clist->first;
  while(ch != 0)
  {
    next_ch = ch->next;
    ch->level_cut_at = -1;
    if (ch->rec == UNRECOGNIZED)      //**** FORCE A NOT RECOGNIZED ON THEM
      if (recognition(image, ch, ch))
        ch->rec = PASSED_RECOGNITION;
      else
        ch->rec = FAILED_RECOGNITION;
    if (ch->rec == FAILED_RECOGNITION)
      chars_added += goodcut_break_character(image, clist, ch);
    ch = next_ch;
  }
  printf("characters added = %d\n",chars_added);
  return chars_added;           // Exit
}

//.....
//.....
//.....                                  MAIN
//.....                                  ....
//.....
//.....
//.....
//.....
// Initialise:
// Should be called prior to using.  This calles all the
// initialisation routines
void 14_initialise()
{
  14_initialise_rot();
  14_initialise_constants();
```

【図192】

```
}

//////////////////////////////////////////////////////////////////
//
void level4(
  cimage& line,                 // The line
  struct Charlist *clist)       // The charlist
{
  int proj_min, proj_max;       // Used to compute size of proj
  int i;

#ifdef VISIBLE
  xs_clr_win();
  int c;
  Cursor1_X = Cursor1_Y = 10;
  line.plot(Cursor1_X, Cursor1_Y);    // Plot input image
  Cursor2_Y = Cursor1_Y + line.sizeR + 20;      // Final output goes at Cursor2
  Cursor2_X = Cursor1_X;
  Cursor3_Y = Cursor2_Y + line.sizeR + 20;
  Cursor3_X = Cursor2_X;
  for (c=0; c<WINDOW_SIZE_X; c++)
    xs_plotxy(c, Cursor3_Y);            // Plot horiz line
  Cursor3_Y = Cursor3_Y + 20;
  Cursor4_Y = Cursor3_Y;
  Cursor4_X = Cursor3_X;
#endif

  if (line.sizeR > MAX_CHAR_HEIGHT)
  {
    printf("\n*** ERROR ***\n");
    printf("*** Line size (=%d) is greater than max line size allocated in multilevel
\n", line.sizeR);
    printf("*** routines (MAX_CHAR_HEIGHT = %d).  Increase MAX_CHAR_HEIGHT.\n", MAX_C
HAR_HEIGHT);
    exit(1);
  }

  // Allocate Proj and VPP
  i = (ROT_MAX_RANGE/ROT_DEG_RESOLUTION)+1;
  proj_min = -Rot[i](line.sizeR);
  proj_max = line.sizeC-proj_min;
  Proj = ivector_ranged(proj_min, proj_max);
  VPP = ivector(line.sizeC);
  FVVP_Zero = 1;                        // Erase the cache

  /*  ..... Program goes here ..... */
  goodcut(line, clist);

#ifdef VISIBLE
  // Print final character list
  l4_plot_charlist(line, clist, Cursor3_X, Cursor3_Y);
#endif
  printf("Displaying final level 4 charlist.  Enter 0 to continue? ");
//  print_asc_charlist(clist.clist->first);     // ******* DELETE ME ********
  i = auto_input_int();

  // Deallocate
  free_ivector(VPP);
  free_ivector_ranged(Proj, proj_min);
}
```

【図195】

```
/*
 * Filename: mymalloc.h
 * Header file for mymalloc.c
 *
 ---------------------------------------------------------------
 */


/* Function Definitions for mymalloc.c */
#ifdef CPP
/* Definition for C++ files */
extern "C" {
  int *ivector(int n);
  void free_ivector(int* v);
  float *fvector(int n);
  void free_fvector(float* v);
  int *ivector_ranged(int n, int n);
  void free_ivector_ranged(int* v, int m);
  int **imatrix(int r, int c);
  void free_imatrix(int** m, int r);
  char **cmatrix(int r, int c);
  void free_cmatrix(char** m, int r);
}
#else
/* Definition for C files */
/* The header file for mallocs */
int *ivector();
void free_ivector();
int *fvector();
void free_fvector();
int *ivector_ranged();
void free_ivector_ranged();
int **imatrix();
void free_imatrix();
char **cmatrix();
void free_cmatrix();
#endif
```

＊　＊

【図209】

```
/* This file prints a character to be recognized on the screen by */
/* calling the routine thieu_recognize() */

void toshi_recognize(size_r, size_c, image)
     int size_r;                        /* The number of rows */
     int size_c;                        /* The number of columns */
     char **image;              /* The image: Bit 1 is the image */
{
  int r,c;

  printf("\nThe character is:\n");
  for (r=0; r<size_r; r++)
  {
    for (c=0; c<size_c; c++)
      if (image[r][c] & 1)
        printf("*");            /* On */
      else
        printf(".");            /* Off */
    printf("\n");
  }
  printf("****** End of character *******\n\n");
}
```

【図244】

```
      colors[x].flags = DoRed | DoGreen | DoBlue;

      colors[x].red = the_color;
      colors[x].green = the_color;
      colors[x].blue = the_color;
      the_color += 500;
    }
    XQueryColors(d,def_colormap,colors,ncolors);

    my_cmap = XCreateColormap(d,DefaultRootWindow(d),DefaultVisual(d,scr),
                              AllocAll);
    XStoreColors(d,my_cmap,colors,ncolors);
  }
*/
```

【図193】

```
/* This program contains common vector and matrix mallocs */
#include<malloc.h>
#include<stdio.h>

/* Routine for error during memory allocation */
void allocation_error(s)
    char* s;                    /* The error text */
{
    int *coredump;

    printf("Memory allocation error %s\nProgram Aborted.\n", s);
    *coredump = 0;
    coredump = 0;
}

/************** IVECTOR ****************/
/* Allocate an integer vector of size [0..n-1] */
int *ivector(n)
    int n;                      /* Size of vector */
{
    int *v;

    v = (int *)malloc((unsigned) n*sizeof(int));
    if (!v) allocation_error("in ivector()");
    return v;
}

/* Free an integer vector v of size [0..n-1] */
void free_ivector(v)
    int *v;                     /* The vector */
{
    free((int *) v);
}

/************** FVECTOR ****************/
/* Allocate an float vector of size [0..n-1] */
float *fvector(n)
    int n;                      /* Size of vector */
{
    float *v;

    v = (float *)malloc((unsigned) n*sizeof(float));
    if (!v) allocation_error("in fvector()");
    return v;
}

/* Free an float vector v of size [0..n-1] */
void free_fvector(v)
    float *v;                   /* The vector */
{
    free((float *) v);
}

/************** RANGED IVECTOR ****************/
/* Allocate a ranged vector v of size [m..n-1] */
int *ivector_ranged(m,n)
    int m,n;                    /* The range */
{
    int *v;

    v = (int *)malloc((unsigned) (n-m)*sizeof(int));
    if (!v) allocation_error("in ivector_ranged()");
    return (v-m);
}

/* Free an integer ranged vector v of size [m..n-1] */
void free_ivector_ranged(v,m)
    int *v;                     /* The vector */
    int m;                      /* The starting index of v */
{
    free((int *) (v+m));
}

/************** IMATRIX ****************/
/* Allocate an integer matrix of size [0..r-1][0..c-1] */
int **imatrix(r, c)
    int r,c;                    /* The size of the matrix */
{
    int** m;
    int i;

    m=(int **)malloc((unsigned) r * sizeof(int *));
    if (!m) allocation_error("1 in imatrix()");
    for (i=0; i<r; i++) {
      m[i] = (int *)malloc((unsigned) c * sizeof(int));
      if (!m[i]) allocation_error("2 in imatrix()");
    }
    return m;
}

/* Free an integer matrix m of size [0..r-1][0..c-1] */
void free_imatrix(m, r)
    int **m;                    /* The matrix */
    int r;                      /* The size (num rows) */
{
    int i;

    for (i=r-1; i>=0; i--)
      free((int *) m[i]);
    free((int **) m);
}

/************** CMATRIX ****************/
/* Allocate an char matrix of size [0..r-1][0..c-1] */
char **cmatrix(r, c)
    int r,c;                    /* The size of the matrix */
{
    char** m;
    int i;

    m=(char **)malloc((unsigned) r * sizeof(char *));
    if (!m) allocation_error("1 in cmatrix()");
    for (i=0; i<r; i++) {
      m[i] = (char *)malloc((unsigned) c * sizeof(char));
      if (!m[i]) allocation_error("2 in cmatrix()");
    }
    return m;
}

/* Free an character matrix m of size [0..r-1][0..c-1] */
void free_cmatrix(m, r)
    char **m;                   /* The matrix */
    int r;                      /* The size (num rows) */
{
    int i;

    for (i=r-1; i>=0; i--)
      free((char *) m[i]);
```

【図196】

```
//
// Filename: outline.c
// Outlining routines, for character segmentation
//
////////////////////////////////////////////////////////////////////////
//
// Bits used in the cimage: ctr. 1 - pixel on
// Bit 0: (1)   ; 0 = Part of an outline
// Bit 1: (2)   ; 1 = Has been extracted for outlined (don't go & outline it!)
// Bit 2: (4)
// Bit 3: (8)
// Bit 4: (16)
// Bit 5: (32)
// Bit 6: (64)
// Bit 7: (128) -

#include"charlist.h"
#include"cimage.h"

#define VERY_SMALL -32000      // A very small integer (for image size)
#define VERY_LARGE 32000       // A very large integer
                               // considered for overlapping case.

// Define directions used by the cursor for outlining
#define LEFT  0
#define UP    1
#define RIGHT 2
#define DOWN  3

// Define global variables for this program.  Used by programs on and
// below find_break_from_outline() routines.  t stands for turtle
int t_left;            // The left boundry column (inclusive)
int t_right;           // The right boundry column (exclusive)
int t_top;             // The row and column of the cursor for outlining
int t_cursorX;
int t_cursorY;
int t_cursorD;         // The direction of the cursor

// Define external variables and routines (for displaying and debugging)
extern int cursor_X;
extern int cursor_Y;
extern print_charsetcimage(cimage& line, struct Character *chr, int x, int y);
extern print_charlist(cimage& line, struct Charlist *clist, int x, int y);


// Test routine: Prints a cimage with outlining and restores image
void print_outlining(cimage& image, int s)
{
  int a,b;

  a = s-75; b = s-75;
  if (a < 0)  a = 0;
  if (b > image.size0)  b = image.size0;

  printf("\n");
  printf("Column Range - %d to %d\n", a, b);
  for (int r=0; r<image.size1; r++)
  {
    for (int c=a; c<b; c++)
    {
      switch (image.pixel[r][c])
      {
        case 0:
          printf(".");
          break;
        case 1:
          printf("0");
          break;
        case 3:
          printf("3");
          break;
        case 7:
          printf("7");
          break;
        case 63:
          printf("*");
          break;
        case 133:
          printf("S");
          image.pixel[r][c] = 1;
          break;
        default:
          printf("?");
          break;
      }
    }
    printf("\n");
  }
  printf("\n");
}


////////////////////////////////////////////////////////////////////////
//
//                      THE NEW ROUTINES.  (Delete all of the above)
//
////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////


////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////
//                      The Outlining Routines
////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////
// Turn left and right inline expansions
inline void t_turn_left() { t_cursorD = (t_cursorD-1) & 3;}
inline void t_turn_right() { t_cursorD = (t_cursorD+1) & 3;}

////////////////////////////////////////////////////////////////////////
// This routine will move the mouse in the direction it is facing
void t_move_forward()
{
  switch(t_cursorD)
    {
      case LEFT:
        t_cursorC--;
        break;
      case RIGHT:
        t_cursorC++;
        break;
      case UP:
        t_cursorR--;
        break;
      case DOWN:
        t_cursorR++;
        break;
    }
```

【図197】

```
}

////////////////////////////////////////////////////////////////////////
// This routine will return what's infront of where the mouse is facing
// It returns a 0 if it is out of range
int t_in_front(image& image)
{
  int oldR, oldC;          // Stores the old direction
  int r;                   // What to return
  oldR = t_cursorR;        // Remembers this position
  oldC = t_cursorC;

  t_move_forward();        // Move into the new space
  // Check range
  if ((t_cursorR < 0) || (t_cursorR >= image.sizeR))
  {
    t_cursorR = oldR;      // Restore cursor position
    t_cursorC = oldC;
    return 0;              // It too high or low,
  }
  if ((t_cursorC < t_start) ||
      (t_cursorC >= t_stop))
  {
    t_cursorR = oldR;      // Restore cursor position
    t_cursorC = oldC;
    return 0;              // It to the left or right of the border
  }
  r = image.pixel(t_cursorR,t_cursorC);
  t_cursorR = oldR;
  t_cursorC = oldC;
  return r;
}

////////////////////////////////////////////////////////////////////////
// Main Outlining Routine ////////////////////////////////////////////
// Note: This routine won't outline a pixel if only one is set.
void extract_outline_char(
    cimage& line,          // Image of the line
    struct CharList* clist,// The character list
    int left_boundry,      // The left boundry of the area to outline
    int right_boundry,     // The right boundry of the area to outline
    int r,                 // The row and column of an 'on' pixel
    int c)                 // On the edge
{
  struct Character *new_char; // Where the new character is going
  int top, bot, left, right; // Used to be put in the new character
  int curve1, curve2;        // Pointers to curve and curve2 of new char.
  int startingR, startingC;  // The starting position
  int not_finished;          // un-set when finished looping
  int i;

  // Set up convient pointers to the new character we're inserting
  new_char = &(clist->chr[clist->size]); // Set new_char to the last character
                                         // on the character list.
  curve1 = new_char->curve1;             // Load in addresses of curves 1 & 2
  curve2 = new_char->curve2;

  // Initialize variables of new outlined character
  for (i=0; i<clist->vector_height; i++) // Initialize curve arrays
  {
    curve1[i] = VERY_LARGE;
    curve2[i] = VERY_SMALL;
  }
  top = bot = r;
  left = right = c;

  // Initialize turtle stuff
  startingR = t_cursorR = r;    // Initialize cursor
  startingC = t_cursorC = c;
  t_cursorD = LEFT;             // Facing left
  t_start = left_boundry;
  t_stop = right_boundry;
  not_finished = 1;
  while(not_finished) // While we haven't returned to starting
  {                   // point
    // Adjust curves for this new position
    line.pixel(t_cursorR)(t_cursorC) = line.pixel(t_cursorR)(t_cursorC) | 2;
                                  // Set second bit to 1 (outlined)
    if (t_cursorC < curve1(t_cursorR))  // If turtle is left of current break
    {
      curve1(t_cursorR) = t_cursorC;    // Move break
      if (t_cursorC < left)             // Update left
        left = t_cursorC;
      if (t_cursorR < top)              // Update top and bot
        top = t_cursorR;
      if (t_cursorR > bot)
        bot = t_cursorR;
    }
    if (t_cursorC > curve2(t_cursorR))  // (Same for right side)
    {                                   // Note: No need to update mins
      curve2(t_cursorR) = t_cursorC-1;  //       and max because the left
                                        //       side (above) will get it
      if (t_cursorC > right)            // Update right
        right = t_cursorC;
    }

    // Check to see if we've entirely circled the image
    if ((t_cursorR == startingR) && (t_cursorC == startingC))
                  // We've returned to the starting point
    {
      i = t_cursorD;                // Remember direction
      t_cursorD = UP;               // Check the space above
      if ((!t_in_front(line) & 7) != 1)  // (If space above is on, and hasn't
      {                                  //  been outlined, don't exit)
        t_cursorD = LEFT;           // Check space left
        if ((!t_in_front(line) & 7) != 1)  // (Likewise for the left space)
        {
          not_finished = 0;         // We're finished, exit the while routine
          break;
        }
      }
      t_cursorD = i;                // Restore direction and continue
    }

    // Move mouse to next position along the edge
    t_turn_left();                  // Check area to left of mouse
    if ((t_in_front(line) & 5) == 1)  // If something there
    {
      t_move_forward();             // Move to left
      continue;                     // And loop again
    }
    t_turn_right();                 // Check area infront of mouse
    if ((t_in_front(line) & 5) == 1)  // If something there
    {
      t_move_forward();             // Move forward
      continue;                     // And loop again
    }
    t_turn_right();                 // Check area to right of mouse
    if ((t_in_front(line) & 5) == 1)  // If something there
    {
      t_move_forward();             // Move to right
      continue;                     // And loop again
    }
```

【図198】

```
    t_turn_right();             // Check area to behind of mouse
    if ((t_in_front(line) & 5) == 1)    // If something there
    {
      t_move_forward();         // Move behind mouse
      continue;                 // And loop again
    }
    not_finished = 0;           // Else it's a one-pixel image.  so exit
    line.pixel[t_cursorR][t_cursorC] = line.pixel[t_cursorR][t_cursorC] | 4;
                                // Make this spot!
  }

  // Load the info into the Charlist
  new_char->top = top;
  new_char->bot = bot+1;        // (Make it exclusive)
  new_char->left = left;
  new_char->right = right+1;    // (Make it exclusive)
  new_char->type = CHARACTER_PURE_OUTLINED;

  charlist_place_new_char(clist);       // Insert the new character

  // Mark the character as removed from the list
  for (r=0; r<line.sizeR; r++)
    for (c=curve1[r]; c < curve2[r]; c++)
      line.pixel[r][c] = line.pixel[r][c] | 4;
}


//////////////////////////////////////////////////////////////////
//////    Main routine to call to add breaks to the image   //////
//////////////////////////////////////////////////////////////////
void do_outlining(              // Returns number of characters added
    cimage& line,               // Image of the line
    struct Charlist* clist,     // The character list
    int start_col,              // The range to search through (inclusive)
    int stop_col)               //    (exclusive)
{
  int r,c;                      // The row and column we're on

  for (c=stop_col-1; c>=start_col; c--) // Search through column range
    for (r=line.sizeR-1; r>=0; r--)     // Scan from bottom to top
      if (((line.pixel[r][c]) & 5) == 1)        // On pixel that hasn't been outlined
        {
//        line.pixel[r][c] = 123;
//        print_outlining(line, 0);
        extract_outline_char(line, clist, start_col, stop_col, r, c);
                                // Outline the character
        }
}
```

【図203】

```
/*
 * The routines in this file determine if a character is recognized or not
 * (for segmentation)
 *
 ------------------------------------------------------------------------
 */
#include"xs.h"                  /* Include Rick's graphics routines */
#include"autoinput.h"

extern int WINDOW_SIZE_Y;

int recognized(a, size_r, size_c) /* Return a 0 for not recognized, 1 for
                                     recognized */
    int **a;                    /* The character matrix a[row][col] */
    int size_r;                 /* The number of rows in a */
    int size_c;                 /* The number of columns in a */
{
  int r,c;                      /* The row and column of the image */

  int dummy;                    /* A dummy variable */


#ifdef VISIBLE
  /* This routine will print the char at the screen's bottom left corner */
  for (r=0-20; r<size_r+20; r++)        /* Loop through the rows */
    for (c=0-20; c<size_c+20; c++)      /* Loop through the cols */
      if ((r >= 0) && (r < size_r) && (c >= 0) && (c < size_c))
        if (a[r][c] != 0)               /* If the pixel is on */
          xs_plotxy(c+20, r + WINDOW_SIZE_Y - size_r - 20);
        else
          xs_unplotxy(c+20, r + WINDOW_SIZE_Y - size_r - 20);
      else
        xs_unplotxy(c+20, r + WINDOW_SIZE_Y - size_r - 20);

  xs_flush();                   /* Flush the display buffer (to make sure
                                   the character gets printed on the screen */
#endif

  printf("Character displayed.  Enter return value (1 = good, 0 = bad)? ");
  dummy = auto_input_int();
  return dummy;
}
```

【図199】

```
//
// Filename: prgmanc.c
// This routine contains stuff to do page segmentation
//
//////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////
//.....................................................................
//

#define PASS_TO_LEVEL_4

#define MAX_CHARS_PER_LINE 250
#define MAX_LINE_HEIGHT 500

#include<stdlib.h>
#include<string.h>
#include"cimages.h"
#include"xs.h"
#include"pymalloc.h"
#include"charlist.h"
#include"autoinput.h"

void process_line(cimage* image, struct Charlist *clist);
void segp_init(cimage* page);
void segp_plot_line(cimage* page, Charlist *clist, int line_top,
                    int line_bot);

int segp_plot_page(cimage* page, int start_row);

#define MDIVISIONS 1    // The number of columns to divide the page into (Original =
 1)
#define CLOSE_OVERLAP 0.3       // What fraction of two lines must overlap (Original
 = 0.3)

#define PERD_DFI_3_MIN_BLOCK_SIZE 0.034         // Minimum line height (inches)

#define RAD (180.0/3.14159)             // Radians to Degrees
#define OVERLAP 1                       // Half the width of the overlap

// Global variables for page size (eliminate border noise
// The line being segmented (so not reallocced)
extern cimage Process_line;

int PAGE_R1;                    // The starting row of the page (inclusive)
int PAGE_R2;                    // The stopping row of the page (exclusive)
int PAGE_R0 = PAGE_R1;          // PAGE_R1
int PAGE_C1;                    // The starting col of the page (inclusive)
int PAGE_C2;                    // The stopping col of the page (exclusive)
int PAGE_C0 = PAGE_C1;

// Define temporary global variables
int WINDOW_SIZE_X;      // Used for plotting
int WINDOW_SIZE_Y;
int cursorX = 10;       // The screen coordinates
int cursorY = 10;

// The 3line variables
int Logic_type;         // 0 for norm, 1 for OR, 2 for AND-OR
int RESOLUTION_3LINE_R_AND;
int RESOLUTION_3LINE_C_AND;
int RESOLUTION_3LINE_R_OR;
int RESOLUTION_3LINE_C_OR;
int RESOLUTION_3LINE_R_CALLED;
int RESOLUTION_3LINE_C_CALLED;

int MAX_CHAR_LEN;       // The maximum character length in pixels

//.....This routine will plot a scaled cimage on the screen .....
void clear_scale_and_plot_cimage(       // The image
  cimage* image)
{
  float scale, scale2;
  int MX, MY, r, c, x, y;
  int i, j, i1, i2, j1, j2;

  // Compute scale to fit image on screen (x and y)
  scale = (float) (WINDOW_SIZE_X-20)/ (float) image.sizeC;
  scale2 = (float) (WINDOW_SIZE_Y-20) / (float) image.sizeR;

  if (scale2 < scale)           // Take the least of the two
    scale = scale2;
  if (scale > 1.0)              // Only shrink the image
    scale = 1.0;

  MY = (int) ((double) image.sizeR * scale);
  MX = (int) ((double) image.sizeC * scale);

  XS_clr_win();                 // Clear the window
  // Draw the image
  for (r=0; r<my; r++)
    for (c=0; c<mx; c++)
      {
        x = (int) ((double) c / scale);
        y = (int) ((double) r / scale);
        if (image.pixel[y][x] != 0)  // border = 10;
          xs_plotxy(10+c, 10+r);
      }

  // Draw Box Around Image
  i1 = 0;
  j1 = MY + 12;
  for (i=0; i< mx+12; i++)
  {
    if ((i+j1) % 2)
      xs_plotxy(i, j1);
    if ((i+j2) % 2)
      xs_plotxy(i, j2);
  }

  i1 = 0;
  i2 = MX + 12;
  for (j=0; j< my+12; j++)
  {
    if ((i+j) % 2)
      xs_plotxy(i1, j);
    if ((i2+j) % 2)
      xs_plotxy(i2, j);
  }

  xs_flush();                   // Refresh the window
}

//.....................................................................
// This routine will create a histogram from cols a to n.
// THIS WILL INSERT A 1 FOR TEXT 0 FOR BLANK!
void make_partial_hist(
  cimage* page,                 // The input image
  int a,                        // The starting and stopping col (inclusive)
  int n,                        //      (exclusive)
  int *a)                       // The output histogram (allocated previously)
{
  int r,c;
```

```
  for (r=0; r<PAGE_R1; r++)              // Zero before PAGE_R1
    a[r] = 0;
  for (r=PAGE_R1; r<PAGE_R2; r++)        // Histogram between R1 and R2
  {
    a[r] = 0;
    for (c=m; c<n; c++)
      if (page.pixel[r][c] != 0)
      {
        a[r]++;
        c = n;                           // Exit this search
      }
  }
  for (r=PAGE_R2; r<page.sizeR; r++)     // Zero after PAGE_R2
    a[r] = 0;

}

//-----------------------------------------------------------
// Given a range (a1,a2) and (b1, b2) will compute fraction of union/intercept
// Required:  a1 <= a2  and  b1 <= b1.  and the ranges overlap.
// If not overlap, will give a negative number
float overlap(
  int a1,                // Range 1
  int a2,
  int b1,                // Range 2
  int b2)
{
  int i1, i2;            // The intercept range
  int u1, u2;            // The union range

  if (a1 < b1) {
    i1 = b1;
    u1 = a1;
  } else {
    i1 = a1;
    u1 = b1;
  }
  if (a2 < b2) {
    i2 = a2;
    u2 = b2;
  } else {
    i2 = b2;
    u2 = a2;
  }
  if (u2 == u1)          // Only you can prevent floating point exceptions!
    return 0.0;

  return ((float) (i2-i1) / (float) (u2-u1));
}



//-----------------------------------------------------------
// This routine will segment an entire page given the page
// It will call ###### routine for each line
// Uses PAGE_R1, etc...  Note: PAGE_R1 must NOT be zero, and PAGE_R2 must NOT
//    be image.sizeR;
//
void segment_page(
    cimage& page,
    struct charlist *clist)     // A characterlist (memory allocated only)
{
  cimage line;           // The line, one at a time
  int *startcol;         // The start of a column (inclusive)
  int *stopcol;          // The stop of a column (exclusive)
  int **hgram;           // Array of histograms: [num][row]
  int **starttext;       // Array of starting text positions
  int **stoptext;        // Array of stopping positions
  int *cursor;           // The cursors for the start and stop texts
  int *size;             // The size of each list
  int *include;          // Include this segment in the next line (0=no, 1=yes)
  int end_of_list;       // Loop termination flag
  int top_div;           // The division with the topmost text
  int range1, range2;    // The text range in the adjacent division (row #'s)
  int crange1, crange2;  // The text range in the current division (row #'s)
  int r1, r2, c1, c2;    // The row and column of line in page
  int min_line_height;   // The minimum line height
  int reject;            // Line rejection flag
  int i,j,a,b,r,c;       // Nice handy variable


  startcol = ivector(NDIVISIONS);
  stopcol = ivector(NDIVISIONS);
  hgram = imatrix(NDIVISIONS, page.sizeR);
  starttext = imatrix(NDIVISIONS, page.sizeR);  // This is overkill!
  stoptext = imatrix(NDIVISIONS, page.sizeR);  // This is overkill!
  cursor = ivector(NDIVISIONS);
  size = ivector(NDIVISIONS);
  include = ivector(NDIVISIONS);

  min_line_height = (int) ((float) page.dpi * PSEG_DPI_2_MIN_BLOCK_SIZE);

  ///////// Step 1:  Define columns and find histograms /////////
  // Find range of columns
  startcol[0] = PAGE_R1;
  for (i=0; i<NDIVISIONS-1; i++)
  {
    c = PAGE_C1 +(PAGE_CD * (i+1)) / NDIVISIONS;
    startcol[i+1] = c - OVERLAP;
    stopcol[i] = c + OVERLAP;
  }
  stopcol[NDIVISIONS-1] = PAGE_C2;
  // Find histograms
  for (i=0; i<NDIVISIONS; i++)           // For each division
  {
    make_partial_histo(page, startcol[i], stopcol[i], hgram[i]);
  }

  ////////////////// Step 2:  Make text area lists ////////////////////
  for (i=0; i<NDIVISIONS; i++)
    // Note: Zeros' are guarenteed on both ends of the histogram
  {
    size[i] = 0;                         // Initialize
    for (j=PAGE_R1+1; j<PAGE_R2; j++)
    {
      if ((hgram[i][j] == 0) && (hgram[i][j-1]) != 0)   // If starting text
        starttext[i][size[i]] = j-1;
      if ((hgram[i][j] != 0) && (hgram[i][j-1]) == 0)   // If stopping text
      {
        stoptext[i][size[i]] = j-1;
        size[i] += 1;
      }
    }
  }

  ////////////////////// Step 3: Find the Lines ///////////////////////////
  for (i=0; i<NDIVISIONS; i++)
    cursor[i] = 0;       // Set all pointers to the top of the list
  end_of_list = 0;

  while(end_of_list == 0)
  {
```

【図201】

```
////////////// Step 3a: Find the highest text
a = ltpage.size.y;          // Going to be high_div
for (i=0; i<NDIVISIONS; i++)
  if (cursor[i] < size[i])          // If valid cursor
  {
    b = startext[i][cursor[i]] + stopext[i][cursor[i]];
                                    // If it's better
    if (b < a)
    {
      a = b;
      top_div = i;
    }
  }

for (i=0; i<NDIVISIONS; i++)        // Initialise include[]
  include[i] = 0;                   // Don't include anything
include[top_div] = 1;               // Include the first one

r1 = startext[top_div][cursor[top_div]];   // Initialise the size of the
r2 = stopext[top_div][cursor[top_div]];    //   image to this segment
c1 = startcol[top_div];
c2 = stopcol[top_div];

////////////// Step 3b: Loop to left and see what columns match
range1 = r1;                        // The adjacent division text range
range2 = r2;
for (i=top_div-1; i>=0; i--)        // Each column to the left
{
  if (cursor[i] < size[i])          // If we're past the end, don't do it
  {
    crange1 = startext[i][cursor[i]];
    crange2 = stopext[i][cursor[i]];
    if (overlap(range1, range2, crange1, crange2) > CLOSE_OVERLAP)
                                    // If overlapping
    {
      include[i] = 1;               // Include this column also
      if (crange1 < c1)             // Adjust range of r1, r2, c1, c2
        c1 = crange1;
      if (crange2 > r2)
        r2 = crange2;
      c1 = startcol[i];
      range1 = crange1;             // Use this for the next range
      range2 = crange2;
    }
  }
}

////////////// Step 3c: Loop to right and see what columns match
range1 = startext[top_div][cursor[top_div]];  // The adjacent division
range2 = stopext[top_div][cursor[top_div]];   // text range.
for (i=top_div+1; i<NDIVISIONS ; i++)         // Each column to the left
{
  if (cursor[i] < size[i])          // If we're past the end, don't do it
  {
    crange1 = startext[i][cursor[i]];
    crange2 = stopext[i][cursor[i]];
    if (overlap(range1, range2, crange1, crange2) > CLOSE_OVERLAP)
                                    // If overlapping
    {
      include[i] = 1;
                                    // Include this column also
      if (crange1 < r1)             // Adjust range of r1, r2, c1, c2
        r1 = crange1;
      if (crange2 > r2)
        r2 = crange2;
      c2 = stopcol[i];
      range1 = crange1;             // Use this for the next range
      range2 = crange2;
    }
  }
}

////////////// Step 3d: Copy the pieces into the image ////////////////////
// Optimise this!  Selectively do fill() and clippage()
line.fill(0, r2-r1, c2-c1);         // Fill up the image with zeros
for (i=0; i<NDIVISIONS; i++)
  if (include[i] == 1)
  {
    for (c=startcol[i]; c<stopcol[i]; c++)
      for (r=startext[i][cursor[i]]; r<stopext[i][cursor[i]]; r++)
        line.pixel[r-r1][c-c1] = (int) page.pixel[r][c];
  }
line.clippage();                    // Remove white space around the line

////////////// Step 3e: Do we reject this image? ////////////////////
                                    // Default is accept
reject = 0;
if ((r2-r1) < min_line_height)
  reject = 1;
//  printf("  image range is %d..%d, %d..%d\n", c1, c2, r1, r2);

////////////// Step 4: Process Line ////////////////////
if (reject == 0)
{
#ifdef VISIBLE
  clear_scale_and_plot_image(line);
#endif
  printf("Enter 0 to run outlining character segmentation, 1 to ignore? ");
  r = auto_input_int();
  if (r == 0)
  {
    process_line(line, clist);
#ifdef DEMO_OUTPUT
    segp_plot_line(page, line, clist, r1, r2);
#endif
  }

////////////// Skip over all the used sections
for (i=0; i<NDIVISIONS; i++)        // Up all the cursors
  if (include[i] == 1)              // Skip over the included areas
    cursor[i] += 1;

////////////// Check if we've gone off the end of the list
end_of_list = 1;
for (i=0; i<NDIVISIONS; i++)
  if (cursor[i] < size[i])          // If the cursor is not at the end...
    end_of_list = 0;                //   ... don't exit

}

free_ivector(include);
free_ivector(size);
free_ivector(cursor);
free_imatrix(stopcext, NDIVISIONS);
free_imatrix(startext, NDIVISIONS);
free_imatrix(hgram, NDIVISIONS);
free_ivector(stopcol);
free_ivector(startcol);
}

////////////////////////////////////////////////////////////////////////
```

【図202】

```
//.....
//.....                    MAIN                    .....
//.....                                            .....
//.....                                            .....
//.....                                            .....
void xs_init()
{
  clmage page;
  struct charlist clist;
  char filename[100], filename2[100];
  int border_ignore;

#ifdef PASS_TO_LEVEL_4
  void id_initialize();        // Initialize layer 4
  id_initialize();
#endif

  Charlist_initialize(&clist, MAX_CHARS_PER_LINE, MAX_LINE_HEIGHT);
  Process_line.fill(0,0,0,2500); // Allocate the memory for the lines

  xs_win_dim(&WINDOW_SIZE_X, &WINDOW_SIZE_Y); // Get the window size

  printf("Input clff filename? /d2/clf/");
  strcpy(filename, "/d2/clf/");
  scanf("%s", filename2);
  strcat(filename, filename2);

  auto_initialize();           // Initialize Auto-input routines

  page.dpi = 300;
// cout << "Input DPI? ";
// cin >> page.dpi;

  printf("Logic Type (0 = None, 1 = OR, 2 = AND-OR, 3 = Scaled)? ");
  Logic_type = auto_input_int();

  if (logic_type == 2)
  {
    RESOLUTION_3LINE_A_AND = RESOLUTION_3LINE_C_AND * 1;
    printf("3-line AND:\n");
    printf("   Enter 3-line constant for Rows? ");
    RESOLUTION_3LINE_A_AND = auto_input_int();
    printf("   Enter 3-line constant for Columns? ");
    RESOLUTION_3LINE_C_AND = auto_input_int();
  }

  if ((logic_type == 1) || (logic_type == 2))
  {
    RESOLUTION_3LINE_A_OR = RESOLUTION_3LINE_C_OR * 1;
    printf("3-line OR:\n");
    printf("   Enter 3-line constant for Rows? ");
    RESOLUTION_3LINE_A_OR = auto_input_int();
    printf("   Enter 3-line constant for Columns? ");
    RESOLUTION_3LINE_C_OR = auto_input_int();
  }

  if (logic_type == 3)
  {
    RESOLUTION_3LINE_A_SCALED = RESOLUTION_3LINE_C_SCALED * 1;
    printf("3-line SCALED:\n");
    printf("   Enter 3-line constant for Rows? ");
    RESOLUTION_3LINE_A_SCALED = auto_input_int();
    printf("   Enter 3-line constant for Columns? ");
    RESOLUTION_3LINE_C_SCALED = auto_input_int();
  }

  MAX_CHAR_LEN = 42;
  printf("Enter MAX_CHAR_LEN? "); // The maximum character length in pixels
  MAX_CHAR_LEN = auto_input_int();

  border_ignore = 1;
// cout << "Input border width to ignore (in pixels)? ";
// cin >> border_ignore;

  printf("Reading in File... ");
  page.readclff(filename);
  printf("Done.\n");

#ifdef DEMO_OUTPUT
  seg_init(page);              // Initialize page printing routines
#endif

// border_ignore = (int) ((float) page.dpi * DPI_3_BORDER_IGNORE);
// printf("Border ignore = %d\n", border_ignore);

  PAGE_A1 = border_ignore;
  PAGE_A2 = page.size.x - border_ignore;
  PAGE_B0 = PAGE_A2 - PAGE_A1;
  PAGE_C1 = page.size.y - border_ignore;
  PAGE_C0 = PAGE_C2 - PAGE_C1;

  // ..... THIS IS JUST FOR TESTING .....
  int i;
  xs_flush();
  printf("Enter 0 to continue? ");
  i = auto_input_int();

  // clear_scale_and_plot_image(page);

  printf("Pausing because page is displayed. Input 0 to segment page? ");
  border_ignore = auto_input_int();

  while(1) {
    segment_page(page, &clist);
    xs_flush();

    printf("Program finished. Enter 0 to run again? ");
    int ttemp;
    ttemp = auto_input_int();
  }

  Charlist_terminate(&clist);
  auto_terminate();
}
```

【図204】

```
//
// Filename: scaleprint.C
// This contains routines to demonstrate the final character segmentation
//
////////////////////////////////////////////////////////////////////////////
//
#include"autoinput.h"
#include"xs.h"
#include"cimages.h"
#include"charlist.h"

#define SCALE 2                 // Set this to adjust page scale
#define LEFT_EDGE 10            // The left edge to plot the characters

extern int WINDOW_SIZE_X;
extern int WINDOW_SIZE_Y;

// Define global static variables (the characters)
int segp_curX;                  // The cursor positions of the
int segp_curY;                  // segmented characters
int segp_bot_row_printed;       // The bottom most row printed in the top 1/2

// Define Colors
#define LINES_R 255
#define LINES_G 0
#define LINES_B 0
#define TEXT_R 0
#define TEXT_G 0
#define TEXT_B 0
#define BOX_R 255
#define BOX_G 0
#define BOX_B 0

////////////////////////////////////////////////////////////////////////////
// This routine will plot a portion of scaled image on the screen
int segp_plot_page(             // Returns the max row printed
  cimage& page,                 // The page to plot
  int start_row)                // The starting row to start printing from
{
  int right_side;               // The border just to the right of the image
  int r,c,x,y;

  x = y = 10;                   //

  if (start_row < 0)            // Force it to be in range
    start_row = 0;

  right_side = 11 + page.sizeC/SCALE;

  xs_set_foreground(LINES_R, LINES_G, LINES_B);
  for (c=0; c<WINDOW_SIZE_X; c++)       // Plot middle line
    xs_plotxy(c, WINDOW_SIZE_Y/2);

  if (start_row < 1)                    // Plot top dotted line
    for (c=9; c<right_side; c++)
      if ((c+1)%2)
        xs_plotxy(c,9);

  for (r=start_row; r<page.sizeR; r+=SCALE)
  {

    xs_set_foreground(LINES_R, LINES_G, LINES_B);
    x = 10;
    if ((y+1)%2)                        // Plot left dotted line
      xs_plotxy(9,y);
    if ((y+right_side)%2)               // Plot right dotted line
      xs_plotxy(right_side,y);

    xs_set_foreground(TEXT_R, TEXT_G, TEXT_B);
    for (c=0; c<page.sizeC; c+=SCALE)
    {
      if (page.pixel[r][c] != 0)
        xs_plotxy(x,y);
      x++;
    }
    y++;
    if (y > WINDOW_SIZE_Y/2 - 10)       // Exit if at middle of page
      return r;
  }

  xs_set_foreground(LINES_R, LINES_G, LINES_B);
  for (c=9; c<right_side; c++)          // Print bottom line
    if ((c+y)%2)
      xs_plotxy(c,y);

  return r;
}


////////////////////////////////////////////////////////////////////////////
// The main routines
////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////
// This routine will initialize the screen for plotting lines on it
void segp_init(cimage& page)
{
  segp_bot_row_printed = -1;    // Make sure it reprints the top part
  xs_clr_win();
  segp_bot_row_printed = segp_plot_page(page, 0);
  segp_curX = LEFT_EDGE;                // Set the new cursor
  segp_curY = WINDOW_SIZE_Y/2 + 20;
  xs_flush();
}


////////////////////////////////////////////////////////////////////////////
// This routine will print another segmented line on the screen,
// updating the screen if necessary
void segp_plot_line(
  cimage& page,                 // The page
  cimage& line,                 // The line we're segmenting
  struct Charlist *clist,       // The character list to print
  int line_top,                 // The top of the line being printed
  int line_bot)                 // The bottom of the line being printed
{
  struct Character *chr;        // The character we're on
  int max_c_height;             // The highest character height
  int new_curx;                 // The new cursor location for next char
  int last_right_edge;          // The right edge of the prev character
  int x,y,r,c;
  int y1,y2;                    // Used to print box around chars

  int color;                    // Test stuff

  color = 0;
  // Find maximum height of the characters in this line and put it in
  // it max_c_height
  max_c_height = 0;
  chr = clist->first;
  while (chr != 0)
  {
```

【図205】

```
    if ((chr->bot - chr->top) > max_c_height)
      max_c_height = chr->bot - chr->top;
    chr = chr->next;
  }

#ifdef VISIBLE
  sswp_bot_row_printed = -1;    // Make sure it erases if visible
#endif

  // Check if we need to replot the entire screen
  if ((line_bot > sswp_bot_row_printed) ||
      (sswp_Curs[ + line.size[0] > WINDOW_SIZE_Y-10))
  {
    printf("Going to scroll... Enter 0 to continue! ");
    scanf("%d", &x);              // Clear the window
    xs_clr_win();
    sswp_bot_row_printed = sswp_plot_page(page, line_top-10);
    sswp_Curs[ = LEFT_EDGE;
    sswp_Curs[ = WINDOW_SIZE_Y/2 - 20;    // Set the new cursor
  }

  // Print the character list
  chr = clist->first;

  while (chr != 0)
  {
    // Check if need to scroll one line
    if (sswp_Curs[ + chr->right - chr->left + 10 > WINDOW_SIZE_X)
    {                             // Scale
      sswp_Curs[ -= max_c_height/SCALE + 10;    // Move down
      sswp_Curs[ = LEFT_EDGE + 50;    // Indent
    }
    y1=10000;                     // The 1
    xs_plot(sswp_Curs[, y , sswp_Curs[);    // Initialize cursor
    xs_set_foreground(TEXT_C, TEXT_B);
    for (i=0; i<line.size[0]; i++)SCALE)
    {
      if (x >= chr->stop) && (x <= chr->bot))
      {
        if (y < y1)               // Find y1, 1st row printed
          y1 = y;
        y2 = y;                   // Set y2 = last row printed
        for (c=chr->left; c < chr->right; c+=SCALE)
        {
          if ((c >= chr->curve1[i]) && (c < chr->curve2[i]))
            if (line.pixel[i][c] & 1)
              xs_plot(x,y);

          x++;
        }
        new_curs[ = x;
        x = sswp_Curs[;
      }
      y++;
    }
    // Draw Box
    xs_set_foreground(BOX_C, BOX_B);
    for (x=sswp_Curs[-2; x<new_curs[+2; x++)
    {
      xs_plot(x, y1-2);
      xs_plot(x, y2+2);
    }
    for (y=y1-2; y<=y2+2; y++)
    {
      xs_plot(sswp_Curs[-2, y);
      xs_plot(new_curs[, y);

//////////////// Get next character and move cursor over

    last_right_edge = chr->right;

    // Get next cursor
    chr = chr->next;

    // Update the cursor (slide to right)
    if (chr)
      if ((chr->left - last_right_edge-2) < 3)
        sswp_Curs[ = new_curs[ + 3;
      else
        sswp_Curs[ = new_curs[ + chr->left - last_right_edge + 2;
  }
  xs_flush();

  sswp_Curs[ += max_c_height/SCALE + 10; // Reposition the cursor for next line
  sswp_Curs[ = LEFT_EDGE;

#ifdef VISIBLE
  printf("Enter 0 to continue! (not auto input!) ");
  scanf("%d", &x);
#endif
}
```

【図206】

```
/* This program will test the charlist */
#include"charlist.h"

void print_charlist(clist)
  struct Charlist *clist;
{
  struct Character *ch;

  printf("Charlist:\n ");
  ch = clist->first;

  while(ch != 0) {
    printf(" Loc: %d: Chars (-,+) = (", ch->loc/8);
    if (ch->prev == 0)
      printf("***");
    else
      printf("%d",ch->prev->top);
    printf(",%d,",ch->top);
    if (ch->next == 0)
      printf("***");
    else
      printf("%d",ch->next->top);
    printf(")\n");

    ch = ch->next;
  }
  printf("End of charlist\n\n");
}

main()
{
  struct Character *ch;
  struct Charlist clist;
  int a[10];                    /* Dummy vector for height */
  int i,p;

  Charlist_initialize(&clist, 10, 10);

  for (i=0; i<5; i++)
  {
    printf("Position of new char %d:? ", i);
    scanf("%d", &p);
    ch = &(clist.chr[clist.size]);
    ch->top = i;
    ch->bot = i+1;
    ch->curve1[i] = ch->curve2[i] = p;
    Charlist_place_new_char(&clist);
    print_charlist(&clist);
  }

  while(1)
  {
    printf("Replace which char? ");
    scanf("%d", &i);

    ch = clist.first;
    while(ch != 0) {
      if (ch->top == i)
        break;
      ch = ch->next;
    }
    if (ch == 0) {
      printf("Character not found.\n");
    } else {
      printf("New position? ");
      scanf("%d", &p);
      ch->curve1[i] = ch->curve2[i] = p;
      Charlist_replace_char(&clist, ch);
      print_charlist(&clist);
    }
  }
}
```

【図220】

```
/*
 * Filename: autoinput.h
 * Header file for autoinput.c
 *
 ----------------------------------------------------------------
 */


/* Function Definitions for mymalloc.c */
#ifdef CPP
/* Definition for C++ files */
extern "C" {
  void auto_initialize();
  int auto_input_int();
  void auto_terminate();
}
#else
/* Definition for C files */
void auto_initialize();
int auto_input_int();
void auto_terminate();
#endif
```

【図223】

```
/*--------------------------------------------------------------*/
void Zerovpp_terminate(nzlist)
     struct NonZeroVppList *nzlist; /* The list */
{
  free_ivector(nzlist->list);   /* Deallocate memory */
  nzlist->allocated = 0;        /* Set so won't be accidently used */
  nzlist->size = 0;
}


/*--------------------------------------------------------------*/
void Zerovpp_add(nzlist, a, b)
     struct NonZeroVppList *nzlist; /* The list */
     int a,b;                   /* Start and stop of the zerovpp */
{
  if (nzlist->size+2 > nzlist->allocated) {
    printf("NonZeroVppList FULL!!!\n");
    exit(1);
  }
  nzlist->list[(nzlist->size)++] = a;
  nzlist->list[(nzlist->size)++] = b;
}
```

【図207】

```
//
// Filename: test_mult.C
// This file is used to test and develop multilevel3.C
////////////////////////////////////////////////////////////////////
//
#include <stdio.h>
#include "cimages.h"
#include "mymalloc.h"
#include "xs.h"
#include "charlist.h"

extern void l4_initialize();
void level4(cimage& line, struct Charlist *clist);

void xs_init()
{
  struct Charlist clist;
  struct Character *ch;
  cimage image;
  int i,j;

  l4_initialize();

  Charlist_initialize(&clist, 10, 300); // 10 chars, 100=max height
  ch = &(clist.chr[clist.size]); // Set new_char to the last character


  ////////////////////////////////////////////////////////////
  // Load the image
  //
  image.fill(1,10,60);
  image.pixel[6][6] = 0;
  image.pixel[7][6] = 0;
  image.pixel[8][6] = 0;
  image.pixel[6][7] = 0;
  image.pixel[7][7] = 0;
  image.pixel[8][7] = 0;
  image.pixel[6][8] = 0;
  image.pixel[7][8] = 0;
  image.pixel[8][8] = 0;
  image.pixel[8][9] = 0;
  image.pixel[7][9] = 0;
  image.pixel[8][9] = 0;
  image.pixel[2][10] = 0;
  image.pixel[3][10] = 0;
  image.pixel[4][10] = 0;
  image.pixel[5][10] = 0;
  image.pixel[6][10] = 0;
  image.pixel[6][10] = 0;
  image.pixel[7][10] = 0;
  image.pixel[8][10] = 0;
  image.pixel[7][11] = 0;
  image.pixel[5][11] = 0;
  image.pixel[4][11] = 0;
  image.pixel[5][11] = 0;
  image.pixel[6][11] = 0;
  image.pixel[6][11] = 0;
  image.pixel[7][11] = 0;
  image.pixel[8][11] = 0;
  image.pixel[0][12] = 0;
  image.pixel[1][12] = 0;
  image.pixel[2][12] = 0;
  image.pixel[3][12] = 0;
  image.pixel[0][13] = 0;
  image.pixel[1][13] = 0;
  image.pixel[2][13] = 0;
  image.pixel[0][14] = 0;
  for (i=1; i<9; i++)
    image.pixel[i][20] = 0;
  for (i=0; i<10; i++)
    image.pixel[i][21] = 0;
  for (i=1; i<9; i++)
    image.pixel[i][22] = 0;
  for (i=27; i<30; i++)
    for (j=2; j<4; j++)
      image.pixel[j][i] = 0;
  for (i=24; i<28; i++)
    for (j=1; j<2; j++)
      image.pixel[j][i] = 0;
  for (i=29; i<31; i++)
    for (j=7; j<9; j++)
      image.pixel[j][i] = 0;

  // Load the character
  ch->top = 0;
  ch->bot = 10;
  ch->left = 0;
  ch->right = 35;
  for (i=ch->top; i<=ch->bot; i++) {
    ch->curve1[i] = ch->left;
    ch->curve2[i] = ch->right;
  }

  Charlist_place_new_char(&clist);

  level4(image, &clist);

  printf("*** End of Program ***\n");
}
```

【図235】

```
/*
 * Filename: mymalloc.h
 * Header file for mymalloc.c
 *
 ---------------------------------------------------------------
 */


/* Function Definitions for mymalloc.c */
#ifdef CPP
/* Definition for C++ files */
extern "C" {
  int *ivector(int n);
  void free_ivector(int* v);
  float *fvector(int n);
  void free_fvector(float* v);
  int *ivector_ranged(int m, int n);
  void free_ivector_ranged(int* v, int n);
  int **imatrix(int r, int c);
  void free_imatrix(int** m, int r);
  char **cmatrix(int r, int c);
  void free_cmatrix(char** m, int r);
}
#else
/* Definition for C files */
/* The header file for mallocs */
int *ivector();
void free_ivector();
int *fvector();
void free_fvector();
int *ivector_ranged();
void free_ivector_ranged();
int **imatrix();
void free_imatrix();
char **cmatrix();
void free_cmatrix();
#endif
```

【図240】

```
/* This file prints a character to be recognized on the screen by */
/* calling the routine thisu_recognize() */

char toshi_recognize(size_r, size_c, image)
     int size_r;                    /* The number of rows */
     int size_c;                    /* The number of columns */
     char **image;              /* The image: Bit 1 is the image */
{
  int r,c;
  char input_string[100];

  printf("\nThe character is:\n");
  for (r=0; r<size_r; r++)
  {
    for (c=0; c<size_c; c++)
      if (image[r][c] & 1)
        printf("*");            /* On */
      else
        printf(".");            /* Off */
    printf("\n");
  }
  printf("***** End of character *******\n\n");
  printf("Enter Character? ");
  scanf("%s", input_string);
  return input_string[strlen(input_string)-1];
}
```

【図208】

```
//
// Filename: test_mult.C
// This file is used to test and develop multilevel2.C
////////////////////////////////////////////////////////////////////////
//
#include <stdio.h>
#include <string.h>
#include "cimages.h"
#include "mymalloc.h"
#include "xs.h"
#include "charlist.h"

extern void l4_initialize();
void level4(cimage& line, struct Charlist *clist);

void xs_init()
{
  struct Charlist clist;
  struct Character *ch;
  cimage image;
  int i,j;

  l4_initialize();

  Charlist_initialize(&clist, 500, 300);        // 10 chars, 100=max height

  char filename1[100];
  char filename2[100];
  printf("Input filename? /d2/tif/");
  scanf("%s", filename1);
  strcpy(filename2, "/d2/tif/");
  strcat(filename2, filename1);
  image.readtiff(filename1);
  image.clipspace();

  ch = &(clist.chr[clist.size]); // Set new_char to the last character

  // Load the character
  ch->top = 0;
  ch->bot = image.sizeR;
  ch->left = 0;
  ch->right = image.sizeC;
  for (i=ch->top; i<ch->bot; i++) {
    ch->curve1[i] = ch->left;
    ch->curve2[i] = ch->right;
  }

  Charlist_place_new_char(&clist);

  level4(image, &clist);

  printf("*** End of Program ***\n");

}
```

【図228】

```
  c1 = get2bytes(f, fp);
  c2 = get2bytes(f, fp);
  return c1 + c2*65536;
}

////////////////////////////////////////////////////////////////////////
// Plot image using Flash Graphics at (xloc, yloc) with color
void cimage::ploc(int x, int y)
{
  int px,py;            // Pixel coords on the screen (upper left)
  char color;

  for (py=0; py<sizeR; py++)
    for (px=0; px<sizeC; px++)
      {
        color = pixel[py][px];
        if (color & 1)
          xs_plotxy(x+px, y+py); // Draw it
      }
};


////////////////////////////////////////////////////////////////////////
// Plot image with a box around it (with color 1!)
void cimage::plotbox(int x, int y)
{
  int px,py;

  for (px=0; px<sizeC+2; px++) {        // Draw horizontal lines
    if ((x+px+y)%2 == 0)
      xs_plotxy(x+px, y);
    if ((x+px+y+sizeR+1)%2 == 0)
      xs_plotxy(x+px, y+sizeR+1);
  }
  for (py=1; py<sizeR+1; py++) {
    if ((x+y+py)%2 == 0)
      xs_plotxy(x, y+py);
    if ((x+sizeC+1+y+py)%2 == 0)
      xs_plotxy(x+sizeC+1, y+py);
  }
  plot(x+1, y+1);       // Plot the image
}
```

```
/*
FILE: util.c

COMMENTS:
      This file contains general utility routines.

      ut_reverse(width,height,buf);
      ut_getbits(x,p,n);
      ut_ascii_bitmap(width,height,buf);

   Usage:

      Just call the routines.
*/

/* ----------------------Includes------------------------ */
#include <stdio.h>
#include <malloc.h>

/* ----------------------Global------------------------- */

/* ----------------------Externs------------------------ */

/*
Routine: ut_reverse(width,height,buf)

Description: Reverse the bits for display format

Return: Nothing

Author: Rick Lee
*/
ut_reverse(width,height,buf)
int width;
int height;
char *buf;
{

   unsigned char val,new_val;
   int x,z,y;

   for(x = 0; x < (width>>3)*height; x++)
   {
      z = 0;
      val = buf[x];
      new_val = 0;
      for(y = 7; y >= 0 ; y--)
      {
         if(ut_getbits(val,z++,1))
            new_val = 0x01 << y | new_val;
      }
      buf[x] = new_val;
   }
}


/*
Routine: ut_getbits(x,p,n)

Description: Check if the bit is turned on or not

Return: Nothing

Author: Rick Lee
*/
ut_getbits(x,p,n)
unsigned char x;      /* The character to slice bit(s) from */
int p;                /* The begining of the bit number */
int n;                /* Number of bits to get */
{
   return(x >> (p+1-n)) & ~(~0 << n);
}

/*
Routine: ut_ascii_bitmap(width,height,buf)

Description: Display the bitmap using ascii characters

Return: Nothing

Author: Rick Lee
*/
ut_ascii_bitmap(width,height,buf)
int width;
int height;
char *buf;
{
   short bit_code = 0xC080;
   short bit_result,t=0;
   short x,y,z,bits;

   for(x = 0; x < height; x++)
   {
      for(y = 0; y < width; y++)
      {
         bits = buf[t++];
         for(z = 0; z < 8; z++)
         {
            bit_result = bits & bit_code;
            if(bit_result != 0)
               printf("1");
            else
               printf("0");
            bit_code = bit_code >> 1;
         }
         bit_code = 0x0080;
      }
      printf("\n");
   }
}
```

```
/*
FILE: xs.C

COMMENTS:
      This file contains routines for displaying 'things'.

      xs_flush();
      xs_plotxy(x,y);
      xs_unplotxy(x,y);
      xs_redisplay();
      xs_dis_bitmap(x,y,width,height,buf);
      xs_clr_win();
      xs_draw_box(x,y,width,height);
      xs_string(x,y,str);
      xs_setcolor(color);
      xs_win_dim(width,height);

  Usage:

      Just call these routines once the hook has been made with the
      display routine.
*/

/* ----------------------Includes------------------------- */
#include <stdio.h>
#include <stdlib.h>
#include <X11/IntrinsicP.h>
#include <X11/StringDefs.h>
#include <Xm/Xm.h>
#include <Xm/DrawingA.h>
#include <Xm/PushB.h>
#include <Xm/Form.h>
#include <X11/Xutil.h>

/* ----------------------Global Var---------------------- */


/* ----------------------Externs------------------------- */
extern Widget parent,              /* The root widget */
              canvas;              /* The display widget */
extern GC line_gc;                 /* The graphic context (colors) */
extern XGCValues gcv;


/************************* ROUTINES ************************/


/* Internal Utility to create/return the pixel of an RGB value */
int xs_get_color(red, green, blue) /* Returns a colormap entry */
  int red, green, blue;        /* From 0 to 255! */
{
  XColor color;
  Colormap cmap;
  Display *dpy = XtDisplay(canvas);

  cmap = DefaultColormap(dpy, DefaultScreen(dpy));

  color.red = red*256;
  color.green = green*256;
  color.blue = blue*256;

  if (XAllocColor(dpy, cmap, &color))
    return (color.pixel);

  printf('Warning: Couldn't allocate requested color\n');
  return BlackPixel(dpy, DefaultScreen(dpy));
}


/* Routine to set printing color */
void xs_set_foreground(red, green, blue)
  int red, green, blue;          /* From 0 to 255! */
{
  XSetForeground(XtDisplay(canvas),
                 line_gc,
                 xs_get_color(red, green, blue));
}


/* Routine to clear window and set background color */
void xs_set_background(red, green, blue)
  int red, green, blue;          /* From 0 to 255! */
{
  XSetBackground(XtDisplay(canvas),
                 line_gc,
                 xs_get_color(red, green, blue));
  xs_clr_win();                  /* Erase it! */
  xs_flush();
}



xs_flush()
{
   /* Flush the queue to X-server */
   XFlush(XtDisplay(canvas));
}


/*
Routine: xs_plotxy(x,y)

Description: Draw a point on the screen

Return: Nothing

Author: Rick Lee
*/
xs_plotxy(x,y)
int x;
int y;
{
   XDrawPoint(XtDisplay(canvas),XtWindow(canvas),line_gc,x,y);
}


/*
Routine: xs_unplotxy(x,y)

Description: UNDraw a point on the screen

Return: Nothing

Author: Rick Lee
*/
xs_unplotxy(x,y)
int x;
int y;
{
   XSetForeground(XtDisplay(canvas),line_gc,gcv.background);
   XDrawPoint(XtDisplay(canvas),XtWindow(canvas),line_gc,x,y);
   XSetForeground(XtDisplay(canvas),line_gc,gcv.foreground);
}
```

【図212】

```
/*
Routine: xs_redisplay()

Description: Redisplay the 'stuff' that was on the window

Return: Nothing

Author: Rick Lee
*/
xs_redisplay()
{
}


/*
Routine: xs_dis_bitmap(x,y,width,height,buf)

Description: Draw the given bitmap

Return: Nothing

Author: Rick Lee
*/
xs_dis_bitmap(x,y,width,height,buf)
{

    Pixmap bitmap,pix;   /* The bitmap for display */

    /* Reverse the bit ordering */
    uc_reverse(width,height,buf);

    /* First create bitmap */
    bitmap = XCreateBitmapFromData(XtDisplay(canvas),
                    RootWindowOfScreen(XtScreen(canvas)),
                    buf,width,height);

    /* Create the pixmap area */
    pix = XCreatePixmap(XtDisplay(canvas),RootWindowOfScreen(XtScreen(canvas)),width,
                DefaultDepthOfScreen(XtScreen(canvas)));

    /* Now copy the data to bitmap */
    XCopyPlane(XtDisplay(canvas),bitmap,pix,line_gc,0,0,
            width,height,0,0,1);

    /* Now copy the pixmap to drawing area */
    XCopyArea(XtDisplay(canvas),pix,XtWindow(canvas),line_gc,0,0,
            width,height,x,y);

    /* Now free the bitmap */
    XFreePixmap(XtDisplay(canvas),bitmap);

    /* Now free the bitmap */
    XFreePixmap(XtDisplay(canvas),pix);

    /* Now flush the queue to X-server */
    XFlush(XtDisplay(canvas));
}


/*
Routine: xs_clr_win()

Description: Clear the screen

Return: Nothing

Author: Rick Lee
*/
xs_clr_win()
{
    XClearArea(XtDisplay(canvas),XtWindow(canvas),0,0,0,0,TRUE);
}


/*
Routine: xs_draw_box(x,y,width,height)

Description: Draw a box on screen

Return: Nothing

Author: Rick Lee
*/
xs_draw_box(x,y,width,height)
int x;
int y;
int width;
int height;
{
    XDrawRectangle(XtDisplay(canvas),XtWindow(canvas),line_gc,
                    x,y,width,height);
}


/*
Routine: xs_string

Description: Draw a string on the screen

Return: Nothing

Author: Rick Lee
*/
xs_string(x,y,str)
int x;
int y;
char *str;
{
    XDrawString(XtDisplay(canvas),XtWindow(canvas),line_gc,
                x,y,str,strlen(str));
}


/*
Routine: xs_setcolor

Description: Set the current color

Return: Nothing

Author: Rick Lee
*/
xs_setcolor(color)
int color;
{
    gcv.foreground = color;
    XSetForeground(XtDisplay(canvas),line_gc,color);
}
```

【図213】

```
/*
Routine: xs_win_dim

Description: Get the size of the window

Return: Nothing

Author: Rick Lee
*/
xs_win_dim(width,height)
int *width;
int *height;
{
   Arg args1[2];
   Dimension win_width,win_height;

   XtSetArg(args1[0],XtNwidth,&win_width);
   XtSetArg(args1[1],XtNheight,&win_height);
   XtGetValues(canvas,args1,2);

   *width = win_width;
   *height = win_height;
}

/*
xs_set_cmap(w)
Widget w;
{
   int ncolors;
   XColor colors[256];
   Colormap my_cmap;
   int count = 1;


   Display *d;
   int scr;
   Colormap def_colormap;
   XColor color;
   int x;
   int the_color = 3000;

   d = XtDisplay(w);
   scr = DefaultScreen(d);
   ncolors = DisplayCells(d,scr);
   def_colormap = DefaultColormap(d,scr);
   for(x = 0; x < ncolors; x++)
   {
      colors[x].pixel = x;
      colors[x].flags = DoRed | DoGreen | DoBlue;

      colors[x].red = the_color;
      colors[x].green = the_color;
      colors[x].blue = the_color;
      the_color += 500;
   }
   XQueryColors(d,def_colormap,colors,ncolors);

   my_cmap = XCreateColormap(d,DefaultRootWindow(d),DefaultVisual(d,scr),
                             AllocAll);
   XStoreColors(d,my_cmap,colors,ncolors);
}
*/
```

【図215】

```
/* If your routines are in C++ then CPP must be defined in make file */
#ifdef CPP
   extern "C" void xs_init();
   extern "C" void xs_flush();
   extern "C" void xs_plotxy(int x,int y);
   extern "C" void xs_unplotxy(int x,int y);
   extern "C" void xs_redisplay();
   extern "C" void xs_dis_bitmap(int x,int y,int width,int height,char *buf);
   extern "C" void xs_clr_win();
   extern "C" void xs_draw_box(int x,int y,int width,int height);
   extern "C" void xs_string(int x,int y,char *str);
   extern "C" void xs_setcolor(int color);
   extern "C" void xs_win_dim(int *width,int *height);
#endif
```

```
/*
FILE: xt.C

COMMENTS:
      This file contains interface to X-display routines.

      main();
      xt_redisplay(w,data,call_data);

   Usage:
      Your initilize routine must be xs_init().
*/

#define DEFAULT_WINDOW_SIZE_HEIGHT 830
#define DEFAULT_WINDOW_SIZE_WIDTH 1130


/* ----------------------Includes----------------------- */
#include <stdio.h>
#include <stdlib.h>
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <Xm/Xm.h>
#include <Xm/DrawingA.h>
#include <Xm/PushB.h>
#include <Xm/Form.h>
#include <X11/Xutil.h>

/* ----------------------Global Var---------------------- */
Widget parent,          /* The root widget */
       canvas;          /* The widget to display */
int start_up = 1;       /* The flag to call init routine */
GC line_gc;             /* Pointer to GC */
XGCValues gcv;          /* The GC values */

/* ----------------------Externs------------------------- */



/*
Routine: xt_redisplay(w,data,call_data)

Description: Expose callback

Return: Nothing

Author: Rick Lee
*/
void xt_redisplay(w,data,call_data)
Widget w;                               /* The widget associated with display */
char *data;                             /* The data  */
XmDrawingAreaCallbackStruct *call_data; /* Not used */
{
   if(start_up)
   {
      start_up = 0;
      xs_init();
   }
}

main(argc,argv)
int argc;
char *argv[];
{
   Arg args[10];
   int n,i;
   int no_set_win_size;         /* = 1 to not set window size */
   int no_set_win_bg;           /* = 1 to not set window background */

#ifdef CPP
   /* Call the routine to setup C++ interface, take out if no C++ interface */
   _main();
#endif

   /* Checks for geometry and background options in command line */
   no_set_win_size = 0;
   no_set_win_bg = 0;
   for (i=0; i<argc; i++) {
     if (strcmp(argv[i],"-geometry") == 0)
       no_set_win_size = 1;
     if (strcmp(argv[i],"-height") == 0)
       no_set_win_size = 1;
     if (strcmp(argv[i],"-width") == 3)
       no_set_win_size = 1;
     if (strcmp(argv[i],"-background") == 0)
       no_set_win_bg = 1;
     if (strcmp(argv[i],"-bg") == 0)
       no_set_win_bg = 1;
   }

   /* Init the Xt function */
   parent = XtInitialize(argv[0],"Pixmap", NULL, 0, &argc,argv);

/*   XtGetApplicationResources(parent, 0, NULL, 0, args, n); */

/*
/*   XSetBackground(XtDisplay(parent), line_gc,
/*                  WhitePixel(XtDisplay(parent),
/*                             DefaultScreen(XtDisplay(parent))));
*/

   /* Create draving area widget */
   canvas = XtCreateManagedWidget("canvas",xmDrawingAreaWidgetClass,
                                  parent,NULL,0);

   /* Set the GC for display routines */
   line_gc = XCreateGC(XtDisplay(canvas),
                       DefaultRootWindow(XtDisplay(canvas)),
                       0, &gcv);

   /* Now add callbacks and event handler */
   XtAddCallback(canvas,XmNexposeCallback,xt_redisplay,NULL);

   /* Set the devault screen size */
   n=0;
   if (no_set_win_size == 0) {
     XtSetArg(args[n], XtNwidth, DEFAULT_WINDOW_SIZE_WIDTH); n++;
     XtSetArg(args[n], XtNheight, DEFAULT_WINDOW_SIZE_HEIGHT); n++;
   }
   if (no_set_win_bg == 0)
     XtSetArg(args[n], XtNbackground, WhitePixel(XtDisplay(parent), DefaultScreen(XtD
isplay(parent)))); n++;
   XtSetValues(canvas, args, n);

   /* Display the widget and enter event loop */
   XtRealizeWidget(parent);


   XtMainLoop();
}
```

【図216】

```
//
// Filename: zerovpp.C
// This routine contains the lines to find blocks that have non-zero vpp
// (level 1 segmentation)
//
////////////////////////////////////////////////////////////////////
//
#include"charlist.h"
#include"cimages.h"

////////////////////////////////////////////////////////////////////
// Scan a line quickly (every 1/4 pixel) looking for an 'on' pixel
int line_scan_quick(    // Returns 1 for pixel found, 0 otherwise
  cimage& line,         // The line to process
  int c)                // The column to scan
{
  int r;

  for (r=0; r<line.sizeR; r+=4)
    if (line.pixel[r][c] & 1)
      return 1;
  return 0;
}


////////////////////////////////////////////////////////////////////
// Scan a line completely looking for an 'on' pixel
int line_scan_complete( // Returns 1 for pixel found, 0 for otherwise
  cimage& line,         // The line
  int c)                // The column to scans
{
  int r;

  for (r=0; r<line.sizeR; r++)
    if (line.pixel[r][c] & 1)
      return 1;
  return 0;
}


////////////////////////////////////////////////////////////////////
// The main routine.  Call this to make the list
void extract_zero_vpp_list(
  cimage& line,         // The image of the line to segment
  struct NonZeroVppList *list)  // A pointer to the list
{
  int c;
  int left, right;

  for (c=0; c<line.sizeC; c+=4)  // Scan the image
  {
    if (line_scan_quick(line, c))  // If found something
    {
      left = right = c;         // Look to the left for the start
      while (--c >= 0)
        if (line_scan_complete(line, c) == 0)
          break;
      left = c+1;
      c = right;                // Keep looking to the right
      while (++c < line.sizeC)
        if (line_scan_complete(line, c) == 0)
          break;
      right = c;
      zerovpp_add(list, left, right);   // Add in the new break
    }
  }
```

【図218】

```
CC=     cc
CPP=    CC
COMPILE_PROFILE = #-p
COMPILE_STYLE = -g # -O
COMPILE_VISIBLE = -DVISIBLE
COMPILE_MAKE_DBASE = #-DMAKE_DATABASE
LDLIBS= -lm
LDLIBSX = -lXm -lXt -lX11 -ll -lmalloc -lPW
CPPFLAGS= -DCPP $(COMPILE_STYLE) $(COMPILE_VISIBLE) $(COMPILE_PROFILE) $(COMPILE_MAKE
_DBASE)
CFLAGS= $(COMPILE_STYLE) $(COMPILE_VISIBLE) $(COMPILE_PROFILE) $(COMPILE_MAKE_DBASE)
OBJECTS_courcut = psegment.o lineprocess.o zerovpp.o charlist.o cimages.o mymalloc.o
xs.o xt.o util.o autoinput.o courcut.o thieu_file.o

all: courcut

courcut: $(OBJECTS_courcut)
        $(CPP) -o courcut $(OBJECTS_courcut) $(LDLIBS) $(LDLIBSX)

autoinput.o: autoinput.c
        $(CC) -c $(CFLAGS) autoinput.c

cimages.o: cimages.C
        $(CPP) -c $(CPPFLAGS) cimages.C

courcut.o: courcut.C
        $(CPP) -c $(CPPFLAGS) courcut.C

lineprocess.o: lineprocess.C
        $(CPP) -c $(CPPFLAGS) lineprocess.C

mymalloc.o: mymalloc.c
        $(CC) -c $(CFLAGS) mymalloc.c

psegment.o: psegment.C
        $(CPP) -c $(CPPFLAGS) psegment.C

scaleprint.o: scaleprint.C
        $(CPP) -c $(CPPFLAGS) scaleprint.C

thieu_file.o: thieu_file.c
        $(CC) -c $(CFLAGS) thieu_file.c

util.o: util.c
        $(CC) -c $(CFLAGS) util.c

xs.o: xs.c
        $(CC) -c $(CFLAGS) xs.c

xt.o: xt.c
        $(CC) -c -DCPP $(CFLAGS) xt.c

zerovpp.o: zerovpp.C
        $(CPP) -c $(CPPFLAGS) zerovpp.C

charlist.o: charlist.h mymalloc.h
combine.o: charlist.h cimages.h
cimages.o: cimages.h xs.h
level4.o: mymalloc.h
lineprocess: charlist.h cimages.h
multilevel.o: charlist.h xs.h mymalloc.h
multilevel2.o: charlist.h cimages.h xs.h mymalloc.h
outline.o: charlist.h cimages.h
psegment.o: cimages.h xs.h mymalloc.h
level3m1.o: charlist.h cimages.h
zerovpp.o: charlist.h cimages.h
```

```
rm_visible:
        rm lineprocess.o psegment.o
```

【図219】

```
/*
 * filename: autoinput.h
 * This file contains the code to input from a filename
 *
 *.....
 */

#include <stdio.h>
#include "autoinput.h"

/* Global variables */

int auto_eof_detected;      /* 1: 0 if eof has been detected */
FILE *auto_fp;              /* The file pointer */
int save_to_file;           /* Saving the input to a file? */
int save_count;             /* Number of inputs before newline */
char filename[100];         /* The filename */

/*.....*/
void auto_initialize()
{
  int not_exit = 1;

  auto_eof_detected = 0;

  while(not_exit)
  {
    printf("Input filename for auto-input? ");
    scanf("%s", filename);

    auto_fp = fopen(filename, "r");

    if (auto_fp == NULL)      /* Read open failed */
    {
      auto_fp = fopen(filename, "w");
      if (auto_fp == NULL)    /* Write open failed */
      {
        printf("Cannot open file %s\n", filename);
      } else {                /* Opened new file */
        printf("%s opened as a new file\n", filename);
        printf("Enter 1 at the next prompt\n");
        save_to_file = 1;
        auto_eof_detected = 1;
        not_exit = 0;
      }
    } else {                  /* Read open succeeded */
      printf("Opening existing file %s\n",filename);
      not_exit = 0;
    }
  }

  printf("Do you wish to save the auto-input into this file? (1 = yes)? ");
  save_to_file = auto_input_int();
  printf("Remember: Change the 1st '1' to a '0' to turn off appending to the file\n");
}
save_count = 0;
}


int auto_input_int()
{
  int i;

  if (auto_eof_detected)
  {
    printf("\n[MANUAL input (-9 = newline, -9999 = exit)] ? ");
    scanf("%d", &i);
    if (i == -9999)
    {
      fclose(auto_fp);
      exit(0);
    }
    if (save_to_file == 1)
    {
      if (i == -9)                    /* Print new lines */
      {
        fprintf(auto_fp, "\n"); /* Print the new lines */
        fprintf(auto_fp, "\n");
        fflush(auto_fp);
        save_count = 0;
        return auto_input_int(); /* And input again */
      }
      fprintf(auto_fp, "%d ", i);
      fflush(auto_fp);
      if (++save_count > 20)
      {
        save_count = 0;
        fprintf(auto_fp, "\n");
      }
    }
    return i;
  }

  if (fscanf(auto_fp, "%d", &i) == EOF)
  {
    printf("\n*** END OF FILE DETECTED ***\n");
    if (save_to_file == 1)
    {
      fclose(auto_fp);
      auto_fp = fopen(filename, "a");
      if (auto_fp == NULL)
      { printf("***** WARNING: CANNOT APPEND TO FILE: %s ***\n", filename);
        save_to_file = 0;
      }
      fprintf(auto_fp, "\n\n");
    }
    auto_eof_detected = 1;
    return auto_input_int();
  }

  printf("%d\n",i);
  return i;
}


void auto_terminate()
{
  fclose(auto_fp);
}
```

【図221】

```
/*
 * Filename: charlist.c
 * This file contains Charlist and Renderer/pplist utility routines
 *
 */

/*-------------------------------------------------------------------*/
/*                  Routines to handle Character lists               */
/*-------------------------------------------------------------------*/
/* By Christopher Sherrick */

#include"charlist.h"
#include"gmalloc.h"

/*-------------------------------------------------------------------*/
/* Initialize and allocate a Characterlist (set up curve pointers) */
/* Initialize routine uses only */
void Charlist_initialize(clist, clist_max, vector_height)
    struct CharList *clist;     /* The Characterlist */
    int clist_max;              /* The max number of characters */
    int vector_height;          /* Height of the line (to allocate curves) */
{
int i;                          /* A looping var */

/* Allocate memory for characters */
clist->chr = (struct Character *) malloc((unsigned) clist_max
                                * sizeof(struct Character));
if (clist->chr) {
    printf("Memory Allocation Error in Charlist_initialize()\n");
    exit(1);
    }
clist->size = 0;            /* Number of characters */
clist->allocated = clist_max; /* Size Allocated */
clist->first = 0;           /* No first character */
clist->vector_height = vector_height;

/* Allocate memory for curves and set pointers */
/* NOTE: To optimize, hardcode the array into the code */
for (i=0; i<clist_max; i++) {
  clist->chr[i].curve1 = ivector(vector_height);
  clist->chr[i].curve2 = ivector(vector_height);
  }
}

/*-------------------------------------------------------------------*/
/* Zero a previously allocated Characterlist */
void Charlist_zero(clist)
  struct CharList *clist;
{
  clist->size = 0;              /* Number of characters */
  clist->first = 0;             /* No first character */
}

/*-------------------------------------------------------------------*/
/* Terminate deallocate the Characterlist */
void Charlist_terminate(clist)
  struct CharList *clist;
{
  int i;

  for (i=clist->allocated-1; i>= 0; i--) { /* Free curves */
    free_ivector(clist->chr[i].curve1);
    free_ivector(clist->chr[i].curve2);
  }

  free((struct Character *) clist->chr);
  clist->size = 0;
  clist->allocated = 0;
  clist->first = 0;
}

/*-------------------------------------------------------------------*/
/* Place a new character in the Charlist. ( chr[size]).
/* Takes the character beyond the Charlist ( chr[size]),
/* recalculates loc, and inserts it into the list */
void Charlist_place_new_char(clist)
  struct CharList *clist;
{
  struct Character *newchar;    /* Pointer to the new character adding */

  newchar = &(clist->chr[clist->size]); /* Set newchar */

  /* Calculate the position of the new character */
Charlist_recalculate_loc(newchar);

  /* Check for full list */          /* Increment the size of the list */
  (clist->size)++;
  if (clist->size >= clist->allocated) /* It too big..*/
  {
    printf("Error in Charlist_place_new_char(), Character list is FULL!\n");
    exit(1);
  }

  /* Place new in the list character */
  Charlist_locate_character(clist, newchar);
}

/*-------------------------------------------------------------------*/
/* This routine repositions a character in the charlist.  It removes
 * the character, recalculates loc and inserts it in the list accordingly
 */
void Charlist_replace_char(clist, ch)
    struct CharList *clist;
    struct Character *ch;
{
  /* Recalculate the position of the character */
  Charlist_recalculate_loc(ch);

  if (clist->size < 2)          /* If it's the only item in the list */
      return;

  /* Remove the character from the list */
  if (ch->prev == 0) {          /* If 1st on list */
    clist->first = clist->first->next;
    clist->first->prev = 0;
  } else {
    ch->prev->next = ch->next;
    if (ch->next != 0)
      ch->next->prev = ch->prev;
  }

  /* Place the character in the list */
  Charlist_locate_character(clist, ch);
}

/*-------------------------------------------------------------------*/
/* Insert a character after doing a cut */
```

【図222】

```
/* This routine will insert the character beyond the charlist (.chr[size]).
 * .loc is recalculated for both ch_after and the new character, however
 * .loc is NOT USED TO PLACE THE CHARACTER.  It will
 * place the new character after ch_after!  After finished, it is an
 * extremely good idea to resort the breaklist!
 */
void Charlist_insert_after_cut(clist, ch_after)
  struct Charlist *clist;
  struct Character *ch_after;   /* Place the new character after this one */
{
  struct Character *newchar;    /* Pointer to the new character adding */

  newchar = &(clist->chr[clist->size]); /* Get newchar */

  /* Calculate the position of the new character */
  Charlist_recalculate_loc(newchar);
  Charlist_recalculate_loc(ch_after);

  /* Check for full list */
  (clist->size)++;              /* Increment the size of the list */
  if (clist->size >= clist->allocated) /* If too big.. */
  {
    printf("Error in Charlist_place_new_char(): Character list is FULL!\n");
    exit(1);
  }

  newchar->prev = ch_after;
  newchar->next = ch_after->next;
  ch_after->next = newchar;
  if (newchar->next != 0)
    newchar->next->prev = newchar;
}

/*-------------------------------------------------------------------*/
/* This routine will remove a character from the character list.
 * Note.  The memory is NOT reclaimed!!!  So don't rely on this too much.
 * Also, prev and next pointers of ch are NOT altered !
 */
void Charlist_remove_char(clist, ch)
    struct Charlist *clist;
    struct Character *ch;
{
  /* Remove the character from the list */
  if (ch->prev == 0) {          /* If 1st on list */
    clist->first = clist->first->next;
    clist->first->prev = 0;
  } else {
    ch->prev->next = ch->next;
    if (ch->next != 0)
      ch->next->prev = ch->prev;
  }
}

/*-------------------------------------------------------------------*/
/* This routine will (re)calculate the position of a character */
void Charlist_recalculate_loc(ch) /* Output is ch->loc */
  struct Character *ch;         /* The character */
{
  int i,loc;
  /* Calculate the position of the new character */
  loc = 0;
  for (i=ch->top; i<ch->bot; i++) /* Position = average position... */
    loc += ch->curve[i] + ch->curve[i+1]; /* ..of the curves */
  loc = loc * 4 / (ch->bot - ch->top);
                                /* Set it */
  ch->loc = loc;
}

/*-------------------------------------------------------------------*/
/* This routine will place a character (that is not in the charlist)
 * into the charlist. */
void Charlist_locate_character(clist, ch)
  struct Charlist *clist;
  struct Character *ch;         /* The character */
{
  int loc;
  struct Character *p;          /* Used for searching for location */
  struct Character *last_p;     /* The character before p */

  /* Is the list empty? */
  if (clist->size == 1)
  {
    ch->next = 0;
    ch->prev = 0;
    clist->first = ch;
    return;
  }

  /* Possible optimization: Don't start from start of list...
     start from where it */     /* Retrieve position */
  loc = ch->loc;

  /* Is it at the beginning of the list? */
  p = clist->first;             /* Load p with the first item on the list */
  if (loc < p->loc)             /* If it's the first on the list */
  {
    p->prev = ch;
    ch->next = clist->first;    /* Set the pointers */
    ch->prev = 0;
    clist->first = ch;          /* Set the 1st on the list to this */
    return;
  }

  while (p->loc <= loc)         /* until find record past where to place */
  {
    last_p = p;                 /* Remember last position */
    p = p->next;                /* Get the next character */
    if (p == 0)                 /* If it's the end of list, insert it there */
      break;
  }
  ch->prev = last_p;
  ch->next = last_p->next;      /* Insert new position after last_p */
  last_p->next = ch;
  if (p!=0)                     /* If newchar isn't at the end of the list */
    p->prev = ch;
}

/*-------------------------------------------------------------------*/
/*-------------------------------------------------------------------*/
/*-------------------------------------------------------------------*/
void zeroVpp_initialize(nzlist, size)
  struct MonteroVpplist *nzlist; /* The list */
  int size;                     /* The number of entries (2x allocated!) */
{
  /* Allocate memory for list */
  size = size*2;                /* Compute memory allocated, etc */
  nzlist->zlist = (vector(size)); /* Allocate memory */
  nzlist->allocated = size;     /* Set max allocated */
  nzlist->size = 0;             /* Set to zero elements */
}
```

【図224】

```
/*
 * Filename: charlist.h
 * Header file for charlist.c
 *
 *.....
 */

/*.....*/
/*. ENUM: CharType:  Different types of characters */
typedef enum {
    CHARACTER_ERASED,
    CHARACTER_ENTIRE_LINE,
    CHARACTER_BAD_APP,
    CHARACTER_PART_OUTLINED,
    CHARACTER_COMBINED
} CharType;

typedef enum {
    PASSED_RECOGNITION,
    FAILED_RECOGNITION,
    UNRECOGNIZED                    /* Recognition hasn't been tried on it */
} CharRecognition;

/*.....*/
/* Struct Character:
 * Contains information about an outlined character in a page
 */
struct Character {
    CharType type;                  /* The type of character */
    CharRecognition rec;            /* If it's recognized */
    char ascii_value;               /* The letter it's recognized as */
    int level_cut_at;               /* The level the character was cut at
                                       (for goodcut) */
    int loc;                        /* The location of the character */
    int top;                        /* The start of the character (top)
                                       (inclusive)*/
    int bot;                        /* The bottom of the character (bottom)
                                       (exclusive) */
    int left;                       /* The left most pixel in this character */
    int right;                      /* The right most pixel in this character */
    int *curve1;                    /* The left side of the character(inclusive)
                                       (-1 = not there) */
    int *curve2;                    /* The Right side of the character (exclusive)
                                       (-1 = not there) */
    struct Character *next;         /* Ptr to next character in the list
                                       (or 0 if last character) */
    struct Character *prev;         /* Ptr to previous character in the list
                                       (or 0 if first character) */
};

/*.....*/
/* Struct Charlist:
 * Contains information about the list of characters */
struct Charlist {
    struct Character *chr;          /* Array: The characters */
    int size;                       /* Number of characters in the chr[] */
    int allocated;                  /* Number of characters allocated */
    int vector_height;              /* The height of the character curves */
    struct Character *first;        /* The first character in linked list */
};

/*.....*/
/*.....                    Zero Vpp Lists Stuff                    .....*/
/*.....*/
/* Struct MonteroVpplist:
 * Contains information about a list for non-zero VPP's: */
struct MonteroVpplist {
    int list;                       /* Array: 2n - start, 2n+1 - stop */
    int allocated;                  /* Memory allocated (2,4,6...) */
    int size;                       /* How many used (2,4,6...) */
};

/*.....*/
/*.....                    Define Routines                    .....*/
/*.....*/

#ifdef CPP

/* Define C++ Routines */
extern "C" {
  void Charlist_initialise(struct Charlist *clist, int clist_max,
                           int vector_height);
  void Charlist_zero(struct Charlist *clist);
  void Charlist_terminate(struct Charlist *clist);
  void Charlist_place_new_char(struct Charlist *clist);
  void Charlist_replace_char(struct Charlist *clist, struct Character *ch);
  void Charlist_insert_after_cut(struct Charlist *clist,
                                 struct Character *ch_after);
  void Charlist_remove_char(struct Charlist *clist, struct Character *ch);
  void Charlist_relocate_loc(struct Character *ch);
  void Charlist_locate_character(struct Charlist *clist, struct Character *ch);
  void Zerovpp_initialise(struct MonteroVpplist *vplist, int size);
  void Zerovpp_terminate(struct MonteroVpplist *vplist);
  void Zerovpp_add(struct MonteroVpplist *vplist, int a, int b);
}

#else

/* Define C Routines */
void Charlist_initialise();
void Charlist_zero();
void Charlist_terminate();
void Charlist_place_new_char();
void Charlist_replace_char();
void Charlist_insert_after_cut();
void Charlist_remove_char();
void Charlist_recalculate_loc();
void Charlist_locate_character();
void Zerovpp_initialise();
void Zerovpp_terminate();
void Zerovpp_add();

#endif
```

【図225】

```
//
//  Filename:  cimages.cpp
//  This file contains a Vlad array image utility functions, for a cimage
//
//////////////////////////////////////////////////////////////////////////
//

// undefine this to make images.c run invisibly (except for error)
//#define PRINT_IMAGES_INFO

#include"cimages.h"
#include"xs.h"

//.....................................................................
//..... CLASS: cimage   (an array of char)..............................
//.....................................................................

cimage::cimage()
{
  sizeR = 0;
  sizeC = 0;
  allocated_sizeR = 0;
  allocated_sizeC = 0;
  dpi = 0;
}

cimage::cimage(int r, int c)
{
  sizeR = 0;                    // so problems won't be caused when
  sizeC = 0;                    //  allocate called deallocate
  allocated_sizeR = 0;
  allocated_sizeC = 0;
  dpi = 0;
  allocate(r,c);
};

cimage::cimage(cimage& img)
{
  allocate(sizeR, sizeC); // Allocate memory for new cimage (loads sizes)
  for (int i=0; i<sizeR; i++)   // Copy old image to new cimage
    for (int j=0; j<sizeC; j++)
      pixel[i][j] = img.pixel[i][j];
};

//////////////////////////////////////////////////////////////////////////
// Delete the cimage
cimage::~cimage()
{
  deallocate();
};

//////////////////////////////////////////////////////////////////////////
// Allocate memory for an cimage from memory
void cimage::allocate(int r, int c)
{
  deallocate();                 // Remove current cimage
  if (r < 0 || c < 0)
  {
    cerr << "Illegal cimage size: " << r << " by " << c << '\n';
    exit(1);
  }
  sizeR = r;                    // Set cimage sizes
  sizeC = c;
  if ((r != 0) && (c != 0))
  {
    pixel = (unsigned char **)malloc((unsigned int) sizeR*sizeof(unsigned char*)); //
Allocate memory
    if (pixel == NULL)   // Allocate memory
    {
      cerr << "Error: Memory Allocation Error (out of memory)\n";
      exit(1);
    }
    for (int i = 0; i<sizeR; i++)
    {
      pixel[i] = (unsigned char *)malloc((unsigned int) sizeC * sizeof(unsigned char)
); // Allocate more mem
      if (pixel[i] == NULL)
      {
        cerr << "Error: Memory Allocation Error (out of memory)\n";
        exit(1);
      }
    }
    allocated_sizeR = sizeR;
    allocated_sizeC = sizeC;
  }
}

//////////////////////////////////////////////////////////////////////////
// Deallocate memory for an cimage from memory
void cimage::deallocate()
{
  if ((allocated_sizeR != 0) && (allocated_sizeC != 0))
  {
    for (int i = 0; i < allocated_sizeR; i++)
      free((char *) pixel[i]);
    if (allocated_sizeR != 0)
      free((char **) pixel);            // Zero out the sizes
    sizeR = 0;
    sizeC = 0;
    allocated_sizeR = 0;
    allocated_sizeC = 0;
  }
}

//////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////////
// This allows allocation of memory from outside the class.  It will
// Allocate memory if it's larger.  If not, it will just adjust the size
void cimage::reallocate(int R, int C)
{
  int new_R, new_C;
  if ((R > allocated_sizeR) || (C > allocated_sizeC))
  {
    new_R = allocated_sizeR;
    new_C = allocated_sizeC;
    if (R > new_R)
      new_R = R;
    if (C > new_C)
      new_C = C;
    allocate(new_R,new_C);      // Re-allocate the image
  }
  sizeR = R;
  sizeC = C;
}

//////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////////
// Make a cimage with a single value: value = value, R and C are the size
void cimage::fill(char value, int R, int C)
{
```

```
  allocate(R,C);                    // Make the new cimage (note size adjusted)
  for (int i=0; i<sizeR; i++)
    for (int j=0; j<sizeC; j++)
      pixel[i][j] = value;
};


////////////////////////////////////////////////////////////////////////
// Copy a section of a cimage into another.  image = reference image
// Range is (R1..R2)  (C1..C2)
void cimage::clip(cimage& image, int R1, int R2, int C1, int C2)
{
  int r,c,t;

  if (R2 < R1)
    {t = R1; R1=R2; R2=t;}              // Swap R1 & R2
  if (C2 < C1)
    {t = C1; C1=C2; C2=t;}              // Swap C1 & C2
  if (R1 < 0)  R1 = 0;                  // Check limits
  if (R2 < 0)  R2 = 0;
  if (C1 < 0)  C1 = 0;
  if (C2 < 0)  C2 = 0;
  if (R1 > image.sizeR)  R1 = image.sizeR;
  if (R2 > image.sizeR)  R2 = image.sizeR;
  if (C1 > image.sizeC)  C1 = image.sizeC;
  if (C2 > image.sizeC)  C2 = image.sizeC;

  allocate(R2-R1, C2-C1);

  dpi = image.dpi;
  for (r=0; r<R2-R1; r++)
    for (c=0; c<C2-C1; c++)
      pixel[r][c] = image.pixel[r+R1][c+C1];
}


////////////////////////////////////////////////////////////////////////
//  Clipspace: reduce the size of an image by removing the blank
// space around it.  This adjusts the image size.
void cimage::clipspace()
{
  int R1, R2, C1, C2;   // The starting and stopping rows of new image
  int i, j;
  struct cimage temp;   // Temporary image

  R1 = sizeR;           // Initialize to extremes
  R2 = 0;
  C1 = sizeC;
  C2 = 0;

  for (i=0; i<sizeR; i++)       // Compute dimensions of new image
    for (j=0; j<sizeC; j++)
      if (pixel[i][j] != 0)
      {
        if (i < R1) R1 = i;     // Left edge?
        if (i > R2) R2 = i;     // Right edge?
        if (j < C1) C1 = j;     // Top edge?
        if (j > C2) C2 = j;     // Left edge?
      }

  if (R1 > R2) R1 = R2 = 0;     // Check for no pixel found
  if (C1 > C2) C1 = C2 = 0;

  temp.allocate(R2-R1+1, C2-C1+1);      // New image is this size
  for (i=R1; i<=R2; i++)                // Copy clipped image
    for (j=C1; j<=C2; j++)
      temp.pixel[i-R1][j-C1] = pixel[i][j];

  allocate(temp.sizeR, temp.sizeC);     // Copy temp image to current image
  for (i=0; i<sizeR; i++)
    for (j=0; j<sizeC; j++)
      pixel[i][j] = temp.pixel[i][j];
}


////////////////////////////////////////////////////////////////////////
// Read in part of a Tiff file
void cimage::readtiff(char *filename;
{
  FILE *tagfile;                // Input file
  long int tagfile_position;    // The our file position (in bytes)
  long int width;               // Width of picture
  long int length;              // Length of picture
  long int black;               // Value of Black pixel
  long int bps;                 // Bytes per strip
  long int raster_offset;       // Start of image data
  long int numtags, tag, type, len, value;      // Tag values
  long int i,j;

  if ((tagfile = fopen(filename, "rb")) == NULL)
  {
    cerr << "\nError in opening " << filename << '\n';
    exit(1);
  }
  tagfile_position = 0;

  // Read in header and make sure it's Intel!
  i = get2bytes(tagfile,tagfile_position);
  if (i != 0x4949)
  {
    cout << "... Error: TIFF File not in Intel Format\n";
    exit(1);
  }

  // Read in Magic number
  i = get2bytes(tagfile,tagfile_position);
  if (i != 0x002A)
  {
    cout << " BAD Magic Number for TIFF file!!!\n";
    exit(1);
  }

  // Find First File directory
  j = get4bytes(tagfile,tagfile_position);
  while(tagfile_position < j)
  {
    i = get2bytes(tagfile,tagfile_position);
  }

  // Tags in Directory
  numtags = get2bytes(tagfile,tagfile_position);

  while (--numtags >= 0) {
    tag = get2bytes(tagfile, tagfile_position);
    type = get2bytes(tagfile, tagfile_position);
    len = get4bytes(tagfile, tagfile_position);
    value = get4bytes(tagfile, tagfile_position);

    if ((len != 1) && ((tag & 32768) == 0))
    {
      cout << "Error: Tag Length != 1)\n";
      exit(1);
```

【図227】

```
while(tagfile_position < raster_offset)
{
  i = getbytes(tagfile,tagfile_position);
}

switch (tag)
{
  case 0x00FE:              // NewSubFileType
    if (value != 0)
    {
      cout << "Can't Deal with Non-Standard NewSubFileTypeYn";
      exit(1);
    }
    break;
  case 0x00FF:              // OldSubFileType
    if (value != 1)
    {
      cout << "Subfile Type not full resolution!\n";
      exit(1);
    }
    break;
  case 0x0100:              // Width
    width = value;
    break;
  case 0x0101:              // Length
    length = value;
    break;
  case 0x0102:              // Bits per sample
    if (value != 1)
    {
      cout << "; Error: Bits/Sample must be 1!";
      exit(1);
    }
    break;
  case 0x0103:              // Compression
    if (value != 1)
    {
      cout << "Compression Used! Error!\n";
      exit(1);
    }
    break;
  case 0x0106:              // Black is
    black = value;
    break;
  case 0x0107:              // Thresholding tag
    if ((value < 1) || (value > 3))
    {
      cout << "Thresholding Tag - Illegal value!\n";
      exit(1)
    }
    break;
  case 0x0111:              // Offset to Raster Data
    raster_offset = value;
    break;
  case 0x0117:
    bpl = value;
    break;
  case 0x011a:
    break;
  case 0x011b:
    break;
  case 0x0128:
    break;
  default:
    if ((value & 32768) == 0) {
      cout << "0x" << hex << value << dec << " Warning: An unsupported tag appears in the TIF file\n";
    }
    break;
  } // switch
} // while

//-------------------------   READ IN FILE   ----------------------------
allocate((int) length, (int) width);

int v;

for (i = 0; i<length; i++)
{
  for (j = 0; j<width; j++)
  {
    if ((j%8) == 0)
    {
      v = getc(tagfile);
      tagfile_position++;
      if (v == EOF)
      {
        cerr << "\n Error: Premature End of File found in cit file!\n";
        exit(1);
      }
    }
    if ((v & 128) == black*128)
      pixel[i][j] = 0;
    else
      pixel[i][j] = 1;
    v = v << 1;
  }
}

fclose(tagfile);

}

////////////////////////////////////////////////////////////////////////////
// Read in a byte.  File position is updated
// eoferror = 1 to error on EOF, 0 not
int cimage::getbyte(FILE *f, long int& file_position, int eoferror)
{
  int c1;
  c1 = getc(f);
  if ((c1 == EOF) && (eoferror == 1))
  {
    cerr << "\n Error: Premature End of File in getbyte!!!\n";
    exit(1);
  }
  file_position++;
  return c1;
}

////////////////////////////////////////////////////////////////////////////
// Read in an integer from a file.  (LSB, MSB)
long int cimage::getbytes(FILE *f, long int& fp)
{
  int c1, c2;
  c1 = getbyte(f, fp, 1);
  c2 = getbyte(f, fp, 1);
  return c1 + c2*256;
}

////////////////////////////////////////////////////////////////////////////
// Read in a long integer
long int cimage::getlbytes(FILE *f, long int& fp)
{
  long int c1, c2;
```

【図229】

```
//
//  Filename:   cimages.h
//  This is the header file for cimages.C
//
//////////////////////////////////////////////////////////////////////
//
#define MEM_DEBUG 1
#include<stream.h>
#include<stdio.h>
#include<stdlib.h>
#include<math.h>

class cimage;           // Forward reference

//**********************************************************************
//*** CLASS: cimage
//**********************************************************************
class cimage
{
public:
  int           sizeR;          // Number of rows (first index of array)
  int           sizeC;          // Number of cols. (second index of array)
  int           allocated_sizeR; // Amount malloced
  int           allocated_sizeC; // Amount malloced
  int           dpi;            // Dots per inch of the image (0 if unknown)
  unsigned char **pixel;        // The binary image (2d array)

  cimage();                     // Initialize
  ~cimage();                    // Destroy
  cimage(int r, int c);         // Initialize with a size
  cimage(cimage& temp);

  void reallocate(int R, int C); // Will allocate memory for an image
  void fill(char value, int R, int C); // Fill image with a constant value
  void clip(cimage& image, int R1, int R2, int C1, int C2);
  void clipspace();                     // Clips zero space surrounding image
  void readtiff(char *filename);  // Read in a tiff image
  void plot(int x, int y);              // Plot binary image
  void plotbox(int x, int y); // Plot with a box around it

private:
  void allocate(int r, int c);      // Allocate memory
  void deallocate();                // Deallocate memory
  int getibyte(FILE *f, long int& file_position, int eoferror);
  long int get2bytes(FILE *f, long int& fp);
  long int get4bytes(FILE *f, long int& fp);
};
```

【図230】

```
for (ccr_start; ccr_stop; i++)
  if (hist[i-cr_start] < cut_value)
  {
    cut_value = hist[i-cr_start];
    cut_at = i;
  }

printf("  Cut at %3d.  (searched %d +/- %3d = (%3d,%3d)\n",
    cut_at-start_col, cut_pos-start_col, COURCUT_DELTA,
    cr_start-start_col, cr_stop-start_col);

// Add the cut and continue
Zerovpp_add(nxtlist, last_cut, cut_at);
last_cut = cut_at;
}
////////////// Insert last cut ////////////////////
Zerovpp_add(nxtlist, last_cut, stop_col);
}

/*.....*/
/* This routine will take each piece from the input image and send it
/* to be processed by the courier cut routine */
void courier_cut(
  struct MonteroVpplist *oldlist,  // The image of the line to segment
  struct MonteroVpplist **nxtlist, // A pointer to the old list (input)
  struct MonteroVpplist *list)     // A pointer to the new list (output)
{
  int i;

  list->size = 0;          // Erase the list;
  for (i=0; i<oldlist->size; i++ 2)
  {
    courcut_process(line, list, oldlist->list[i], oldlist->list[i+1]);
  }
}

////////////////////////////////////////////////////////////////////////
// Filename: courcut.C
// This routine cuts courier chars (output from level 1)
// (level 1 segmentation).
//
////////////////////////////////////////////////////////////////////////
//
#include"charlist.h"
#include<images.h>

/* Define globals */
int COURCUT_AVG_WIDTH;
int COURCUT_DELTA;

////////////////////////////////////////////////////////////////////////
// This routine will take care of the non-zero vpp list
void courcut_process(
  cimage cline,            // The line
  struct MonteroVpplist *nxtlist, // A pointer to the old list (input)
  int start_col,           // Range to proces
  int stop_col)
{
  int cr_start, cr_stop;   // Region for the new cut
  int last_cut;            // Position of last cut
  int num_cuts;            // The number of cuts to make
  int cut_value, cut_at;   // The value and location of the likely cut
  int hist[100];           // Histogram ******* CHANGE 100 *******
  int width;               // Width of the line
  int cut_pos;             // Approx position to cut at
  int cut_cnt              // The cut number being processed
  int i,c,i;

  /* Calculate num_cuts */
  width = stop_col - start_col; // The width
  if (width <= COURCUT_AVG_WIDTH+COURCUT_DELTA)
    num_cuts = 1;          // It's only one character
  else {
    // Note: The +2 then /2 will do the division with rounding
    num_cuts = (int)((float) width / (float) COURCUT_AVG_WIDTH + 0.5);
  }
  printf("courCut Block width= %d => %d characters\n", width, num_cuts);

  last_cut = start_col;
  for (cut_cnt=1; cut_cnt<num_cuts; cut_cnt++)
  {
    ////////////////////// Break up the region ////////////////////////////
    cut_pos = (width * cut_cnt) / num_cuts + start_col;
    cr_start = cut_pos - COURCUT_DELTA;
    cr_stop = cut_pos + COURCUT_DELTA;
    if (cr_start < start_col)  // Make sure it doesn't go off edges
      cr_start = start_col;
    if (cr_stop > stop_col)
      cr_stop = stop_col;

    // Make histogram over this range
    for (c=cr_start; c<cr_stop; c++)
    {
      hist[c-cr_start] = 0;
      for (i=0; i<line.size.h; i++)
        if (line.pixel[i][c] & 1)
          (hist[c-cr_start])++;
    }

    // Find minimum in histogram
    cut_value = 9999;
    cut_at = (cr_start+cr_stop)/2;
```

```
//
// Filename: lineprocess.C
// This routine contains stuff to do character segmentation
//
//////////////////////////////////////////////////////////////////
//

#include"charlist.h"
#include"cimage.h"
#include"xs.h"
#include"mymalloc.h"
#include"autoinput.h"

#define PASS_TO_TOSHI           // Call Toshi's routines (but ignore output)

#define PASS_TO_LEVEL_2 // Defines when runs level 2
//#define PASS_TO_LEVEL_3  // Define if to run level 3
#define PASS_TO_LEVEL_4 // Call the level 4 routines
//#define COMBINING     // Call the combining routines
//#define DASH_CHARS    // Call the routines to cut dashed characters
//#define PRINT_FINAL_CHARLIST

// Declare external routines
void courrier_cut(cimage& line, struct NonZeroVppList *oldlist,
          struct NonZeroVpplist *list);
void extract_zero_vpp_list(cimage& line, struct NonZeroVppList *list);
#ifdef PASS_TO_TOSHI
extern "C" char toshi_recognize(int r, int c, char** image);
#endif

// Define global variables
int cursor_X, cursor_Y; // Screen coordinates

extern int WINDOW_SIZE_X;
extern int WINDOW_SIZE_Y;

extern int Logic_type;          // 0 for none, 1 for OR, 2 for AND-OR

// The external Variables
cimage Process_line;    // The line being segmented (so not remalloced)

//////////////////////////////////////////////////////////////////
// Misc Utility function
int imin(               // Returns minimum of two values
  int a,
  int b)
{
  if (a>b)
    return b;
  return a;
}

//////////////////////////////////////////////////////////////////
// Print a nonzerovpplist on the screen
int print_nzvpplist(            // Returns height printed
    cimage& line,
    struct NonZeroVppList *nzlist,
    int x,                      // The position
    int y)
{
  int x_left;           /* Left column in nzvpp list */
  int y_start;          /* The starting y */
  int i,r,c;

  x_left = x-nzlist->list[0];
  y_start = y;

  for (i=0; i<nzlist->size; i+=2)
  {
    if (x_left+i*3+nzlist->list[i+1] > WINDOW_SIZE_X - 20) /* Wrap? */
    {
      x_left = x - i*3 - nzlist->list[i];
      y += line.sizeR+10;
    }
    for (r=0; r<line.sizeR; r++)
      for (c=nzlist->list[i]; c<nzlist->list[i+1]; c++)
        if (line.pixel[r][c] & 1)
          xs_plotxy(x_left + c + i*3, y + r);
    for (r=0; r<line.sizeR; r++)        // Plot double line
    {
      xs_plotxy(x_left+i*3+nzlist->list[i]-1, y+r);
      xs_plotxy(x_left+i*3+nzlist->list[i+1], y+r);
    }
  }
  return line.sizeR + (y - y_start);
}

//////////////////////////////////////////////////////////////////
// Print a nonzerovpplist on the screen with a character
int print_nzvpplist_char(       // Returns height printed
    cimage& line,
    struct NonZeroVppList *nzlist,
    char What_Char[],           // The corresponding characters
    int x,                      // The position
    int y)
{
  int x_left;           /* Left column in nzvpp list */
  int y_start;          /* The starting y */
  char dummy_string[10];
  int i,r,c;

  x_left = x-nzlist->list[0];
  y_start = y;

  for (i=0; i<nzlist->size; i+=2)
  {
    if (x_left+i*3+nzlist->list[i+1] > WINDOW_SIZE_X - 20) /* Wrap? */
    {
      x_left = x - i*3 - nzlist->list[i];
      y += line.sizeR+15;
    }
    for (r=0; r<line.sizeR; r++)
      for (c=nzlist->list[i]; c<nzlist->list[i+1]; c++)
        if (line.pixel[r][c] & 1)
          xs_plotxy(x_left - c + i*3, y + r);
    for (r=0; r<line.sizeR; r++)        // Plot double line
    {
      xs_plotxy(x_left+i*3+nzlist->list[i]-1, y+r);
      xs_plotxy(x_left+i*3+nzlist->list[i+1], y+r);
    }

    // Plot the string
    if (What_Char[i/2] != 0)
    {
      dummy_string[0] = What_Char[i/2];
      dummy_string[1] = 0;
      xs_string(x_left+i*3+(nzlist->list[i]+nzlist->list[i+1])/2-2,
                y+r+9, dummy_string);
    }
  }
  return line.sizeR + (y - y_start);
}
```

【図232】

```
//.....
//.....
//.....                    MAIN
//.....
//.....
//.....
void process_line(
    charpt input_line)          // The image of the line to segment
{
  charpt *line_to_process;      // The line to process
  struct NonZeroVppList *nlist; // The nonzero vpp list
  struct NonZeroVppList *clist; // The courier cut list
  char What_Char[200];          // Which character is in the NZVPP list

// No Logic (INSERT ROUTINES HERE IF NEEDED)
  line_to_process = &(input_line);          // The default

  struct NonZeroVppList masternlist;
  struct NonZeroVppList masterclist, 100];  // *** DELETE ***/
  zerovpp_initialize(&masternlist, 100);
  nlist = &masternlist;          // ***** ADD THIS HIGHER!!! *****/

  struct NonZeroVppList ccurcutnlist;  // *** DELETE ***/
  zerovpp_initialize(&ccurcutnlist,100);
  cclist = &ccurcutnlist;        // ***** ADD THIS HIGHER!!! *****/

#ifdef VISIBLE
  xs_clr_win();
  cursor_x = 10;                 // The screen coordinates
  cursor_y = 10;

  (*line_to_process).plc(cursor_x,cursor_y);
  xs_flush();
  cursor_y += (*line_to_process).size_x + 10;
#endif

  ////////// Insert breaks caused by zero values of VPP
  nlist->size = 0; // Zero the list
  extract_zero_vpp_list((*line_to_process), nlist);

  ////////// Insert breaks from the courier cut routine
  courier_cut((*line_to_process), nlist, cclist);

#ifdef PASS_TO_TCSH1
  int size_r, size_c;
  int r,c,i;
  char *p;
  int row_top, row_bot;
  int col_left, col_right;

  for (i=0; i<cclist->size; i++)
  {
    col_left = cclist->list[i];
    col_right = cclist->list[i+1];

    // Find top and bot rows (skip over blank lines)
    row_top = row_bot = -1;
    for (r=0; (r < (*line_to_process).size) && (row_top == -1); r++)
      for (c=col_left; c<col_right; c++)
        if ((*line_to_process).pixel[r][c] & 1) == 1)
          row_top = r;
    for (r=(*line_to_process).size-1; (r >= 0) && (row_bot == -1); r--)
      for (c=col_left; c<col_right; c++)
        if (((*line_to_process).pixel[r][c] & 1) == 1)
          row_bot = r+1;

    // Copy image into matrix D and call tcshl_recognize
    if ((row_bot != -1) && (row_top != -1))
    {
      size_r = row_bot - row_top;
      size_c = col_right - col_left;
      p = CmatrixMalloc(size_r, size_c);
      for (r=0; r<size_r; r++)
        for (c=0; c<size_c; c++)
          p[r][c] = (*line_to_process).pixel[r+row_top][c+col_left];
      What_Char[i/2] = tcshl_recognize(size_r, size_c, p);
      free_cmatrix(p, size_r);
    }
  }
#endif

#ifdef VISIBLE
  #ifdef PASS_TO_TCSH1
    cursor_y -=10-print_nzvpplist_char(*line_to_process).cclist,
                                  What_Char,cursor_x,cursor_y);
  #else
    cursor_y -=10-print_nzvpplist(*line_to_process).cclist,cursor_x,cursor_y);
  #endif
  xs_flush();
#endif

  printf("Displaying final segmentation.  Enter 0 to continue) ");
  int dummy = auto_input_int();
  dummy = dummy;

  zerovpp_terminate(&masternlist);    // *** DELETE ****
  zerovpp_terminate(&ccurcutnlist);   // *** DELETE ****
}
```

[図233]

```
/* This program contains common vector and matrix mallocs */
#include<malloc.h>
#include<stdio.h>

/* Routine for error during memory allocation */
void allocation_error(s)
     char* s;                       /* The error text */
{
  int *coredump;

  printf("Memory allocation error %s\nProgram Aborted.\n", s);
  *coredump = 0;
  coredump = 0;
}

/************ IVECTOR ***************/
/* Allocate an integer vector of size [0..n-1] */
int *ivector(n)
     int n;                         /* Size of vector */
{
  int *v;

  v = (int *)malloc((unsigned) n*sizeof(int));
  if (!v) allocation_error("in ivector()");
  return v;
}

/* Free an integer vector v of size [0..n-1] */
void free_ivector(v)
     int *v;                        /* The vector */
{
  free((int *) v);
}

/************ FVECTOR ***************/
/* Allocate an float vector of size [0..n-1] */
float *fvector(n)
     int n;                         /* Size of vector */
{
  float *v;

  v = (float *)malloc((unsigned) n*sizeof(float));
  if (!v) allocation_error("in fvector()");
  return v;
}

/* Free an float vector v of size [0..n-1] */
void free_fvector(v)
     float *v;                      /* The vector */
{
  free((float *) v);
}

/************ RANGED IVECTOR ***************/
/* Allocate a ranged vector v of size [m..n-1] */
int *ivector_ranged(m,n)
     int m,n;                       /* The range */
{
  int *v;

  v = (int *)malloc((unsigned) (n-m)*sizeof(int));
  if (!v) allocation_error("in ivector_ranged()");
  return (v-m);
}

/* Free an integer ranged vector v of size [m..n-1] */
void free_ivector_ranged(v,m)
     int *v;                        /* The vector */
     int m;                         /* The starting index of v */
{
  free((int *) (v+m));
}

/************ IMATRIX ***************/
/* Allocate an integer matrix of size [0..r-1][0..c-1] */
int **imatrix(r, c)
     int r,c;                       /* The size of the matrix */
{
  int** m;
  int i;

  m=(int **)malloc((unsigned) r * sizeof(int *));
  if (!m) allocation_error("1 in imatrix()");
  for (i=0; i<r; i++) {
    m[i] = (int *)malloc((unsigned) c * sizeof(int));
    if (!m[i]) allocation_error("2 in imatrix()");
  }
  return m;
}

/* Free an integer matrix m of size [0..r-1][0..c-1] */
void free_imatrix(m, r)
     int **m;                       /* The matrix */
     int r;                         /* The size (num rows) */
{
  int i;

  for (i=r-1; i>=0; i--)
    free((int *) m[i]);
  free((int **) m);
}

/************ CMATRIX ***************/
/* Allocate an char matrix of size [0..r-1][0..c-1] */
char **cmatrix(r, c)
     int r,c;                       /* The size of the matrix */
{
  char** m;
  int i;

  m=(char **)malloc((unsigned) r * sizeof(char *));
  if (!m) allocation_error("1 in cmatrix()");
  for (i=0; i<r; i++) {
    m[i] = (char *)malloc((unsigned) c * sizeof(char));
    if (!m[i]) allocation_error("2 in cmatrix()");
  }
  return m;
}

/* Free an character matrix m of size [0..r-1][0..c-1] */
void free_cmatrix(m, r)
     char **m;                      /* The matrix */
     int r;                         /* The size (num rows) */
{
  int i;

  for (i=r-1; i>=0; i--)
    free((char *) m[i]);
```

【図236】

```
//
// Filename: pagmant.C
// This routine contains stuff to do page segmentation
//
//////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////////
//..........................................................................
//


#define MAX_CHARS_PER_LINE 250
#define MAX_LINE_HEIGHT 500

#include<stdlib.h>
#include<string.h>
#include"xs.h"
#include"xsmalloc.h"
#include"charlist.h"
#include"autolayout.h"

void process_line(cimage& image);
void segp_fill(cimage& page);
void segp_plot_lines(cimage& page, cimage& line, Charlist *clist, int line_top,
                     int line_bot);

int segp_plot_page(cimage& page, int start_row);

#define NDIVISIONS 15   // The number of columns to divide the page into  (Original
 15)
#define CLOSE_OVERLAP 0.3      // What fraction of the lines must overlap (Original
 = 0.3)

#define PERC_BPT_2_MIN_BLOCK_SIZE 0.034        // Minimum line height (inches)

#define RAD (180.0/3.14159)            // Radians to Degrees
#define OVERLAP 3                      // Half the width of the overlap

// Define external functions
extern cimage *process_line();   // The line being segmented (so not reallocated)
extern void loc_load_font(char *filel); // Loads in the font

// Global variables for page size (eliminate border noise)

int PAGE_R1;                   // The starting row of the page (inclusive)
int PAGE_R2;                   // The stopping row of the page (exclusive)
int PAGE_RD;                   // PAGE_R2 - PAGE_R1
int PAGE_C1;                   // The starting col of the page (inclusive)
int PAGE_C2;                   // The stopping col of the page (exclusive)
int PAGE_CD;                   // PAGE_C2 - PAGE_C1

// Define temporary global variables
int WINDOW_SIZE_X;             // Used for plotting
int WINDOW_SIZE_Y;
int cursorX = 10;              // The screen coordinates
int cursorY = 10;

extern int COURIOR_AVG_WIDTH;
extern int COURIOR_DELTA;

//.......... This routine will plot a scaled cimage on the screen ..........
void clear_scale_and_plot_cimage(
  cimage& image)               // The image
{
  float scale, scale2;

  int MX, MY, r, c, x, y;
  int i, j, i1, i2, j1, j2;

  // Compute scale to fit image on screen (x and y)
  scale = (float) (WINDOW_SIZE_X-20)/ (float) image.sizeC;
  scale2 = (float) (WINDOW_SIZE_Y-20) / (float) image.sizeR;

  if (scale2 < scale)          // Take the least of the two
    scale = scale2;
  if (scale > 1.0)             // Only shrink the image
    scale = 1.0;

  MY = (int) ((double) image.sizeR * scale);
  MX = (int) ((double) image.sizeC * scale);

  XS_clr_win();                // Clear the window
  // Draw the image
  for (r=0; r<MY; r++)
  for (c=0; c<MX; c++)
  {
    x = (int) ((double) c / scale);
    y = (int) ((double) r / scale);
    if (image.pixel[y][x] != 0)
      XS_plotxy(10+c, 10+r);  // Border = 10;
  }

  // Draw Box Around Image
  j1 = 8;
  j2 = MY + 12;
  for (i=8; i< MX+12; i++)
  {
    if (((i+1) % 2)
      XS_plotxy(i, j1);
    if ((i%2) == 0)
      XS_plotxy(i, j2);
  }

  i1 = 8;
  i2 = MX + 12;
  for (j=8; j< MY+12; j++)
  {
    if (((j+1) % 2)
      XS_plotxy(i1, j);
    if ((i2+j) % 2)
      XS_plotxy(i2, j);
  }

  XS_flush();                  // Refresh the window
}

//..........................................................................
// This routine will create a histogram from cols m to n.
// THIS WILL INSERT A 1 FOR TEXT 0 FOR BLANK.
void make_partial_hist(
  cimage& page,                // The input image
  int m,                       // The starting and stopping col (inclusive)
  int n,                       //                (exclusive)
  int *a)                      // The output histogram (allocated previously)
{
  int r,c;

  for (c=0; c<PAGE_R1; c++)    // Zero before PAGE_R1
    a[c] = 0;
  for (c=PAGE_R1; c<PAGE_R2; c++)      // Histogram between R1 and R2
  {
    a[c] = 0;
```

【図237】

```
    for (c=0; c<h; c++)
      { if (page.plist[r][c] != 0)
          {
            a[r]++;
            c = h;
          }                                   // Exit this search
      }
    for (r=PAGE_H2; r<page.ulsAR; r++)        // Zero after PAGE_H2
      a[r] = 0;
  }

//.....
// Given a range (a1,a2) and (b1, b2) will compute fraction of union/intercept
// required:  a1 <= a2  and  b1 <= b2,  and the ranges overlap.
// If not overlap, will give a negative number
float overlap(
  int a1,                                     // Range 1
  int a2,
  int b1,                                     // Range 2
  int b2)
{
  int i1, i2;                                 // The intercept range
  int u1, u2;                                 // The union range

  if (a1 < b1) {
    i1 = b1;
    u1 = a1;
  } else {
    i1 = a1;
    u1 = b1;
  }
  if (a2 < b2) {
    i2 = a2;
    u2 = b2;
  } else {
    i2 = b2;
    u2 = a2;
  }
  if (u2 == u1)                               // Only you can prevent floating point exceptions!
    return 0.0;
  return (float) (i2-i1) / (float) (u2-u1);
}

//.....
// This routine will segment an entire page given the page
// It will call 0(1)0 routine for each line.
// Uses PAGE_H2, etc...  Note: PAGE_H2 must NOT be zero, and PAGE_H2 must NOT
//    be image.elsAR;
//
void segment_page(
  charpos_page
  struct charlist *clist)     // A characterlist (memory allocated only)
{
  cinage line;                // The line, one at a time
  int *startcol;              // The start of a column (inclusive)
  int *stopcol;               // The stop of a column (exclusive)
  int **hgram;                // Array of histograms: [num][row]
  int **startText;            // Array of starting text positions
  int **stopText;             // Array of stopping positions
  int *cursor;                // The cursors for the start and stop texts
  int *size;                  // The size of each list
  int *include;               // Include this segment in the next line (0-no, 1-yes)
  int end_of_list;            // Loop termination flag
  int top_div;                // The division with the topmost text
  int range1, range2;         // The text range in the adjacent division (row 1's)
  int crange1, crange2;       // The text range in the current division (row 1's)
  int r1, r2, c1, c2;         // The row and column of line in page
  int min_line_height;        // The minimum line height
  int reject;                 // Line rejection flag
  int i,j,e,b,r,c;            // Nice hardy variable

  startcol = ivector(NDIVISIONS);
  stopcol = ivector(NDIVISIONS);
  hgram = imatrix(NDIVISIONS, page.ulsAR);   // This is overkill!
  startText = imatrix(NDIVISIONS, page.ulsAR); // This is overkill!
  stopText = imatrix(NDIVISIONS);
  size = ivector(NDIVISIONS);
  include = ivector(NDIVISIONS);

  min_line_height = (int) ((float) page.dpi * PSEG_DPI_2_MIN_BLOCK_SIZE);

  ////////// Step 1:  Define columns and find histograms //////////
  // Find range of columns
  startcol[0] = PAGE_M1;
  for (i=0; i<NDIVISIONS-1; i++)
  {
    c = PAGE_C1 +(PAGE_C0 * (i+1)) / NDIVISIONS;
    startcol[i+1] = c - OVERLAP;
    stopcol[i] = c + OVERLAP;
  }
  stopcol[NDIVISIONS-1] = PAGE_C2;
  // Find histograms
  for (i=0; i<NDIVISIONS; i++)                // For each division
  {
    make_partial_histo(page, startcol[i], stopcol[i], hgram[i]);
  }

  ////////////////// Step 2:  Make text area lists //////////////////
  for (i=0; i<NDIVISIONS; i++)
    // Note: Zeros' are guaranteed on both ends of the histogram
  {
    size[i] = 0;
    for (j=PAGE_M1-1; j<PAGE_M2; j++)         // Initialize.
    {
      if ((hgram[i][j] == 0) && (hgram[i][j+1]) != 0)   // If starting text
        startText[i][size[i]] = j+1;
      if ((hgram[i][j] != 0) && (hgram[i][j+1]) == 0)   // If stopping text
        {
          stopText[i][size[i]] = j+1;
          size[i]++;
        }
    }
  }

  ////////////////// Step 3:  Find the lines //////////////////
  for (i=0; i<NDIVISIONS; i++)
    cursor[i] = 0;                            // Set all pointers to the top of the list
  end_of_list = 0;

  while(end_of_list == 0)
  {
    ////////// Step 3a:  Find the highest text
    r = page.ulsAR;                           // Going to be high_div
    for (i=0; i<NDIVISIONS; i++)
      if (cursor[i] < size[i])                // If valid cursor
      {
        b = startText[i][cursor[i]] + stopText[i][cursor[i]];
```

```
    if (b < a)                      // If it's better
    {
      a = b;
      top_div = i;
    }
  }

for (i=0; i<NDIVISIONS; i++)        // Initialize include[]
  include[i] = 0;                   // Don't include anything
include[top_div] = 1;               // Include the first one

r1 = starttext[top_div][cursor[top_div]];   // Initialize the size of the
r2 = stoptext[top_div][cursor[top_div]];    //   cimage to this segment
c1 = startcol[top_div];
c2 = stopcol[top_div];

////////////// Step 3b: Loop to left and see what columns match
range1 = r1;                        // The adjacent division text range
range2 = r2;
for (i=top_div-1; i>=0; i--)        // Each column to the left
{
  if (cursor[i] < size[i])          // If we're past the end, don't do it
  {
    crange1 = starttext[i][cursor[i]];
    crange2 = stoptext[i][cursor[i]];
    if (overlap(range1, range2, crange1, crange2) > CLOSE_OVERLAP)
                                    // If overlapping
    {
      include[i] = 1;                       // Include this column also
      if (crange1 < r1)             // Adjust range of  r1, r2, c1, c2
        r1 = crange1;
      if (crange2 > r2)
        r2 = crange2;
      c1 = startcol[i];
      range1 = crange1;             // Use this for the next range
      range2 = crange2;
    }
  }
}

////////////// Step 3c: Loop to right and see what columns match
range1 = starttext[top_div][cursor[top_div]];  // The adjacent division
range2 = stoptext[top_div][cursor[top_div]];   // text range.
for (i=top_div+1; i<NDIVISIONS ; i++)       // Each column to the left
{
  if (cursor[i] < size[i])          // If we're past the end, don't do it
  {
    crange1 = starttext[i][cursor[i]];
    crange2 = stoptext[i][cursor[i]];
    if (overlap(range1, range2, crange1, crange2) > CLOSE_OVERLAP)
                                    // If overlapping
    {
      include[i] = 1;                       // Include this column also
      if (crange1 < r1)             // Adjust range of  r1, r2, c1, c2
        r1 = crange1;
      if (crange2 > r2)
        r2 = crange2;
      c2 = stopcol[i];
      range1 = crange1;             // Use this for the next range
      range2 = crange2;
    }
  }
}


////////////// Step 3d: Copy the pieces into the cimage ////////////////////
// Optimize this:  Selectively do fill() and clipspace()

  line.fill(0, r2-r1, c2-c1);       // Fill up the image with zeros
  for (i=0; i<NDIVISIONS; i++)
    if (include[i] == 1)
    {
      for (c=startcol[i]; c<stopcol[i]; c++)
        for (r=starttext[i][cursor[i]]; r<stoptext[i][cursor[i]]; r++)
          line.pixel[r-r1][c-c1] = (int) page.pixel[r][c];
    }
  line.clipspace();             // Remove white space around the line

  ////////////////// Step 3e: Do we reject this image? ////////////////////
  reject = 0;                   // Default is accept
  if ((r1-r1) < min_line_height)
    reject = 1;
//  printf("  cimage range is %d..%d, %d..%d\n", c1, c2, r1, r2);

  ////////////////// Step 4f: Process line ////////////////////
  if (reject == 0)
  {
#ifdef VISIBLE
    clear_scale_and_plot_cimage(line);
#endif
    printf("Enter 0 to run character segmentation, 1 to ignore? ");
    r = auto_input_int();
    if (r == 0)
    {
      process_line(line);
//      segp_plot_line(page, line, clist, r1, r2);
    }
  }

  ////////////////// Skip over all the used sections
  for (i=0; i<NDIVISIONS; i++)      // Up all the cursors
    if (include[i] == 1)
      cursor[i] += 1;               // Skip over the included areas

  ////////// Check if we've gone off the end of the list
  end_of_list = 1;
  for (i=0; i<NDIVISIONS; i++)
    if (cursor[i] < size[i])        // If the cursor is not at the end...
      end_of_list = 0;              // ... don't exit

}

  free_ivector(include);
  free_ivector(size);
  free_ivector(cursor);
  free_imatrix(stoptext, NDIVISIONS);
  free_imatrix(starttext, NDIVISIONS);
  free_imatrix(hgram, NDIVISIONS);
  free_ivector(stopcol);
  free_ivector(startcol);
}

//*************************************************************
//*************************************************************
//************                MAIN                 ************
//*************************************************************
//*************************************************************
void xs_init()
{
  cimage page;
  struct Charlist clist;
  char filename[100], filename1[100], filename2[100];
```

【図239】

```
  int border_ignore;
  int i;

  Charlist_initialize(&clist, MAX_CHARS_PER_LINE, MAX_LINE_HEIGHT);
  Process_line.fill(0,40,3560); // Allocate the memory for the lines

  xs_win_dim(&WINDOW_SIZE_X, &WINDOW_SIZE_Y);  // Get the window size

  printf("Input tiff filename? /d2/tif/");
  strcpy(filename1, "/d2/tif/toshii.tif");
  scanf("%s", filename1);
  strcpy(filename, "/d2/tif/");
  strcat(filename, filename1);

  auto_initialize();            // Initialize Auto-Input routines

  printf("Enter courcut average width? ");
  COURCUT_AVG_WIDTH = auto_input_int();

  printf("Enter courcut delta? ");
  COURCUT_DELTA = auto_input_int();

  page.dpi = 300;

  border_ignore = 1;
//  cout << "Input border width to ignore (in pixels)? ";
//  cin >> border_ignore;

  printf("Reading in File... ");
  page.readtiff(filename);
  printf("Done.\n");

//  segp_init(page);                    // Initialize page printing routines

//  border_ignore = (int) ((float) page.dpi * DPI_2_BORDER_IGNORE);
//  printf("Border ignore = %d\n", border_ignore);

  PAGE_R1 = border_ignore;              // Define page size
  PAGE_R2 = page.sizeR - border_ignore;
  PAGE_RD = PAGE_R2 - PAGE_R1;
  PAGE_C1 = border_ignore;
  PAGE_C2 = page.sizeC - border_ignore;
  PAGE_CD = PAGE_C2 - PAGE_C1;


  // ************ THIS IS JUST FOR TESTING ********************
  xs_flush();
  printf("Enter 0 to continue? ");
  i = auto_input_int();

  // clear_scale_and_plot_cimage(page);

  printf("Pausing because page is displayed. Input 0 to segment page? ");
  border_ignore = auto_input_int();

  while(1) {
    segment_page(page, &clist);
    xs_flush();

    printf("Program finished. Enter 0 to run again? ");
    int tttemp;
    tttemp = auto_input_int();
  }

  Charlist_terminate(&clist);
  auto_terminate();
}
```

[図241]

```
/*
FILE: util.C

COMMENTS:
      This file contains general utility routines.

      ut_reverse(width,height,buf);
      ut_getbits(x,p,n);
      ut_ascii_bitmap(width,height,buf);

   Usage:

      Just call the routines.
*/

/* ---------------------Includes---------------------- */
#include <stdio.h>
#include <malloc.h>

/* ---------------------Global------------------------ */

/* ---------------------Externs----------------------- */

/*
Routine: ut_reverse(width,height,buf)

Description: Reverse the bits for display format

Return: Nothing

Author: Rick Lee
*/
ut_reverse(width,height,buf)
int width;
int height;
char *buf;
{

   unsigned char val,new_val;
   int x,z,y;

   for(x = 0; x < (width>>3)*height; x++)
   {
      z = 0;
      val = buf[x];
      new_val = 0;
      for(y = 7; y >= 0 ; y--)
      {
         if(ut_getbits(val,z++,1))
            new_val = 0x01 << y | new_val;
      }
      buf[x] = new_val;
   }
}


/*
Routine: ut_getbits(x,p,n)

Description: Check if the bit is turned on or not

Return: Nothing

Author: Rick Lee
*/
ut_getbits(x,p,n)
unsigned char x;      /* The character to slice bit(s) from */
int p;                /* The begining of the bit number */
int n;                /* Number of bits to get */
{
   return(x >> (p+1-n)) & ~(~0 << n);
}

/*
Routine: ut_ascii_bitmap(width,height,buf)

Description: Display the bitmap using ascii characters

Return: Nothing

Author: Rick Lee
*/
ut_ascii_bitmap(width,height,buf)
int width;
int height;
char *buf;
{
   short bit_code = 0x0080;
   short bit_result,t=0;
   short x,y,z,bits;

   for(x = 0; x < height; x++)
   {
      for(y = 0; y < width; y++)
      {
         bits = buf[t++];
         for(z = 0; z < 8; z++)
         {
            bit_result = bits & bit_code;
            if(bit_result != 0)
               printf("1");
            else
               printf("0");
            bit_code = bit_code >> 1;
         }
         bit_code = 0x0080;
      }
      printf("\n");
   }
}
```

```
/*
FILE: xs.C

COMMENTS:
      This file contains routines for displaying 'things'.

      xs_flush();
      xs_plotxy(x,y);
      xs_unplotxy(x,y);
      xs_redisplay();
      xs_dis_bitmap(x,y,width,height,buf);
      xs_clr_win();
      xs_draw_box(x,y,width,height);
      xs_string(x,y,str);
      xs_setcolor(color);
      xs_win_dim(width,height);

   Usage:

      Just call these routines once the hook has been made with the
      display routine.
*/

/* ----------------------Includes---------------------- */
#include <stdio.h>
#include <stdlib.h>
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <Xm/Xm.h>
#include <Xm/DrawingA.h>
#include <Xm/PushB.h>
#include <Xm/Form.h>
#include <X11/Xutil.h>

/* ----------------------Global Var---------------------- */


/* ----------------------Externs---------------------- */
extern Widget parent,           /* The root widget */
      canvas;                   /* The display widget */
extern GC line_gc;
extern XGCValues gcv;

xs_flush()
{
   /* Flush the queue to X-server */
   XFlush(XtDisplay(canvas));
}


/*
Routine: xs_plotxy(x,y)

Description: Draw a point on the screen

Return: Nothing

Author: Rick Lee
*/
xs_plotxy(x,y)
int x;
int y;
{
   XDrawPoint(XtDisplay(canvas),XtWindow(canvas),line_gc,x,y);
}

/*
Routine: xs_unplotxy(x,y)

Description: UNDraw a point on the screen

Return: Nothing

Author: Rick Lee
*/
xs_unplotxy(x,y)
int x;
int y;
{
   XSetForeground(XtDisplay(canvas),line_gc,gcv.background);
   XDrawPoint(XtDisplay(canvas),XtWindow(canvas),line_gc,x,y);
   XSetForeground(XtDisplay(canvas),line_gc,gcv.foreground);
}


/*
Routine: xs_redisplay()

Description: Redisplay the 'stuff' that was on the window

Return: Nothing

Author: Rick Lee
*/
xs_redisplay()
{
}


/*
Routine: xs_dis_bitmap(x,y,width,height,buf)

Description: Draw the given bitmap

Return: Nothing

Author: Rick Lee
*/
xs_dis_bitmap(x,y,width,height,buf)
{

   Pixmap bitmap,pix;   /* The bitmap for display */

   /* Reverse the bit ordering. */
   ut_reverse(width,height,buf);

   /* First create bitmap */
   bitmap = XCreateBitmapFromData(XtDisplay(canvas),
                                  RootWindowOfScreen(XtScreen(canvas)),
                                  buf,width,height);

   /* Create the pixmap area */
   pix = XCreatePixmap(XtDisplay(canvas),RootWindowOfScreen(XtScreen(canvas)),width,h
eight,
                       DefaultDepthOfScreen(XtScreen(canvas)));

   /* Now copy the data to bitmap */
   XCopyPlane(XtDisplay(canvas),bitmap,pix,line_gc,0,0,
              width,height,0,0,1);

   /* Now copy the pixmap to drawing area */
```

【図243】

```
    XCopyArea(XtDisplay(canvas),pix,XtWindow(canvas),line_gc,0,0,
              width,height,x,y);

    /* Now free the bitmap */
    XFreePixmap(XtDisplay(canvas),bitmap);

    /*  Now free the bitmap */
    XFreePixmap(XtDisplay(canvas),pix);

    /* Now flush the queue to X-server */
    XFlush(XtDisplay(canvas));
}



/*
Routine: xs_clr_win()

Description: Clear the screen

Return: Nothing

Author: Rick Lee
*/
xs_clr_win()
{
   XClearArea(XtDisplay(canvas),XtWindow(canvas),0,0,0,0,TRUE);
}


/*
Routine: xs_draw_box(x,y,width,height)

Description: Draw a box on screen

Return: Nothing

Author: Rick Lee
*/
xs_draw_box(x,y,width,height)
int x;
int y;
int width;
int height;
{
   XDrawRectangle(XtDisplay(canvas),XtWindow(canvas),line_gc,
                  x,y,width,height);
}


/*
Routine: xs_string

Description: Draw a string on the screen

Return: Nothing

Author: Rick Lee
*/
xs_string(x,y,str)
int x;
int y;
char *str;
{
   XDrawString(XtDisplay(canvas),XtWindow(canvas),line_gc,
               x,y,str,strlen(str));
}
```

```
/*
Routine: xs_setcolor

Description: Set the current color

Return: Nothing

Author: Rick Lee
*/
xs_setcolor(color)
int color;
{
   gcv.foreground = color;
   XSetForeground(XtDisplay(canvas),line_gc,color);
}

/*
Routine: xs_win_dim

Description: Get the size of the window

Return: Nothing

Author: Rick Lee
*/
xs_win_dim(width,height)
int *width;
int *height;
{
   Arg args1[2];
   Dimension win_width,win_height;

   XtSetArg(args1[0],XtNwidth,&win_width);
   XtSetArg(args1[1],XtNheight,&win_height);
   XtGetValues(canvas,args1,2);

   *width = win_width;
   *height = win_height;
}

/*
xs_set_cmap(w)
Widget w;
{
   int ncolors;
   XColor colors[256];
   Colormap my_cmap;
   int count = 1;


   Display *d;
   int scr;
   Colormap def_colormap;
   XColor color;
   int x;
   int the_color = 3000;

   d = XtDisplay(w);
   scr = DefaultScreen(d);
   ncolors = DisplayCells(d,scr);
   def_colormap = DefaultColormap(d,scr);
   for(x = 0; x < ncolors; x++)
   {
      colors[x].pixel = x;
```

[図246]

```
/*
FILE: xt.C

DOCUMENTS:
      This file contains interface to X-display routines.

      main();
      xt_redisplay(w,data,call_data);

   Usage:
      Your initilize routine must be xs_init().
*/

/* ----------------------Includes----------------------- */
#include <stdio.h>
#include <stdlib.h>
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <Xm/Xm.h>
#include <Xm/DrawingA.h>
#include <Xm/PushB.h>
#include <Xm/Form.h>
#include <X11/Xutil.h>

/* ----------------------Global Var--------------------- */
Widget parent,          /* The root widget */
       canvas;          /* The widget to display */
int start_up = 1;       /* The flag to call init routine */
GC line_gc;             /* Pointer to GC */
XGCValues gcv;          /* The GC values */

/* ----------------------Externs------------------------ */



/*
Routine: xt_redisplay(w,data,call_data)

Description: Expose callback

Return: Nothing

Author: Rick Lee
*/
void xt_redisplay(w,data,call_data)
Widget w;                               /* The widget associated with display */
char *data;                             /* The data  */
XmDrawingAreaCallbackStruct *call_data; /* Not used */
{
   if(start_up)
   {
     start_up = 0;
     xs_init();
   }
}

main(argc,argv)
int argc;
char *argv[];
{
   Arg args1[10];           /* Arguments to X-server */
   int n=0;                 /* Number of arguments */
   int background,          /* Default background and foreground color */
       foreground;

#ifdef CPP
   /* Call the routine to setup C++ interface, take out if no C++ interface */
   _main();
#endif

   /* Init the Xt function */
   parent = XtInitialize(argv[0],"Pixmap",NULL,0,&argc,argv);


   /* Create drawing area widget */
   n=0;
   canvas = XtCreateManagedWidget("canvas",xmDrawingAreaWidgetClass,
                                  parent,args1,n);

   /* Get the background and foreground color */
   XtSetArg(args1[0],XtNforeground,&foreground);
   XtSetArg(args1[1],XtNbackground,&background);
   XtGetValues(canvas,args1,2);

   /* Now add callbacks and event handler */
   XtAddCallback(canvas,XmNexposeCallback,xt_redisplay,NULL);

   /* Display the widget and enter event loop */
   XtRealizeWidget(parent);

   /* Set the GC for display routines */
   gcv.foreground = foreground;
   gcv.background = background;
   line_gc = XCreateGC(XtDisplay(canvas),
                       DefaultRootWindow(XtDisplay(canvas)),
                       GCForeground | GCBackground,&gcv);

   XtMainLoop();
}
```

【図247】

```
//
// Filename: zerovpp.C
// This routine contains the lines to find blocks that have non-zero vpp
// (level 1 segmentation)
//
////////////////////////////////////////////////////////////////////
//
#include"charlist.h"
#include"cimages.h"

////////////////////////////////////////////////////////////////////
// Scan a line quickly (every 1/4 pixel) looking for an 'on' pixel
int line_scan_quick(    // Returns 1 for pixel found, 0 otherwise
  cimage& line,         // The line to process
  int c)                // The column to scan
{
  int r;

  for (r=0; r<line.sizeR; r+=4)
    if (line.pixel[r][c] & 1)
      return 1;
  return 0;
}


////////////////////////////////////////////////////////////////////
// Scan a line completely looking for an 'on' pixel
int line_scan_complete( // Returns 1 for pixel found, 0 for otherwise
  cimage& line,         // The line
  int c)                // The column to scana
{
  int r;

  for (r=0; r<line.sizeR; r++)
    if (line.pixel[r][c] & 1)
      return 1;
  return 0;
}


////////////////////////////////////////////////////////////////////
// The main routine.  Call this to make the list
void extract_zero_vpp_list(
  cimage& line,         // The image of the line to segment
  struct NonZeroVppList *list)  // A pointer to the list
{
  int c;
  int left, right;

  for (c=0; c<line.sizeC; c+=4)  // Scan the image
  {
    if (line_scan_quick(line, c))  // If found something
    {
      left = right = c;         // Look to the left for the start
      while (--c >= 0)
        if (line_scan_complete(line, c) == 0)
          break;
      left = c+1;
      c = right;                // Keep looking to the right
      while (++c < line.sizeC)
        if (line_scan_complete(line, c) == 0)
          break;
      right = c;
      ZeroVpp_Add(list, left, right);   // Add in the new break
    }
  }
```

フロントページの続き

(72)発明者 シン・ヤン ワング
アメリカ合衆国 カリフォルニア州 92626, コスタメサ ブルマン ストリート 3188 キヤノン インフォメーション システムズ インク. 内

(72)発明者 メザードゥ アール. バエズィー
アメリカ合衆国 カリフォルニア州 92626, コスタメサ ブルマン ストリート 3188 キヤノン インフォメーション システムズ インク. 内

(72)発明者 クリストファー エー. シェリック
アメリカ合衆国 カリフォルニア州 92626, コスタメサ ブルマン ストリート 3188 キヤノン インフォメーション システムズ インク. 内